User's Manual

Code Generator Tool

User's Manual: RX API Reference

Target Device
RX Family

Renesas Electronics
www.renesas.com

Rev.1.00 July 2018

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.
   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, 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.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

How to Use This Manual

Readers	The target readers of this manual are the application system engineers who use the Code Generator and need to understand its function.	
Purpose	The purpose of this manual is to explain the user for understanding and using the Code Generator functions. We aim to help their system development including their hardware and software.	
Organization	This manual can be broadly divided into the following units. 1.GENERAL 2.OUTPUT FILES 3.API FUNCITONS	
How to Read This Manual	It is assumed that the readers of this manual have general knowledge of electricity, logic circuits, and microcontrollers.	
Conventions	Deata significance:	Higher digits on the left and lower digits on the right
	Active low representation:	$\overline{\text{XXX}}$ (overscore over pin or signal name)
	Note:	Footnote for item marked with Note in the text
	Caution:	Information requiring particular attention
	Remark:	Supplementary information
	Numeric representation:	Decimal ... XXXX Hexadecimal ... 0xXXXX

TABLE OF CONTENTS

1. GENERAL

Code Generator Tool is a software tool that automatically generates device drivers.
This manual explains about.
This manual gives Output file and API functions.

1.1 Overview

Code Generator tool enables you to output the pin assignment of the microcontroller (device pin list and device top view), and the source code (device driver programs, C source files and header files) necessary to control the peripheral functions (clock generator, port functions, etc.) provided by the microcontroller by configuring various information using the GUI.

1.2 Features

The Code Generator tool has the following features.

- Code generating function

  The Code Generator can output not only device driver programs in accordance with the information configured using the GUI, but also a build environment such as sample programs containing main functions and link directive files.

- Reporting function

  The user can change default names assigned to the files output by the Code Generator and the API functions contained in the source code.

- Renaming function

  The user can change default names assigned to the files output by the Code Generator and the API functions contained in the source code.

- User code protective function

  The user can add user's original source code to each API function. When user generated the device driver programs again by the Code Generator, user's source code within this comment is protected.

  [Comment for user source code descriptions]
  /* Start user code. Do not edit comment generated here */
  /* End user code. Do not edit comment generated here */

2. OUTPUT FILES

This appendix describes the files output by the Code Generator.

2.1 Description

Below is a list of output file files by the Code Generator

Table 2.1 Output File List (1/10)

Peripheral Function	File Name	API Function Name	output (*1)
Common CCRX	r_cg_main.c	main	A
		R_MAIN_UserInit	A
	r_dbsct.c	-	-
	r_cg_intprg.c	r_undefined_exception	A
		r_privileged_exception	A
		r_floatingpoint_exception	A
		r_access_exception	A
		r_nmi_exception	A
		r_brk_exception	A
		r_reserved_exception	A
		r_icu_group_n_interrupt	A
	r_cg_resetprg.c	PowerON_Reset	A
		PowerON_Reset_PC	A
	r_cg_sbrk.c	-	-
	r_cg_vecttbl.c	-	-
	r_cg_sbrk.h	-	-
	r_cg_stacksct.h	-	-
	r_cg_vect.h	-	-
	r_cg_hardware_setup.c	HardwareSetup	A
		R_Systeminit	A
	r_cg_macrodriver.h	-	-
	r_cg_userdefine.h	-	-
Common IAR(ICCRX)	r_cg_main.c	main	A
		R_MAIN_UserInit	A
	r_cg_intprg.c	r_brk_exception	A
		r_icu_group_n_interrupt	A
		_NMI_handler	A
	r_cg_systeminit.c	R_Systeminit	A
		_low_level_init	A
	r_cg_macrodriver.h	-	-
	r_cg_userdefine	-	-

Table 2.2 Output File List (2/10)

Peripheral Function	File Name	API Function Name	output (*1)
Common GNU(GCCRX)	r_cg_main.c	main	A
		R_MAIN_UserInit	A
	r_cg_vector_table.c	r_undefined_exception	A
		r_reserved_exception	A
		r_nmi_exception	A
		r_brk_exception	A
		r_icu_group_n_interrupt	A
	r_cg_reset_program.asm	-	-
	r_cg_interrupt_handlers.h	-	-
	r_cg_hardware_setup.c	HardwareSetup	A
		R_Systeminit	A
	r_cg_macrodriver.h	-	-
	r_cg_userdefine.h	-	-
Clock generation circuit	r_cg_cgc.c	R_CGC_Create	A
		R_CGC_Set_ClockMode	M
	r_cg_cgc_user.c	R_CGC_Create_UserInit	M
		r_cgc_oscillation_stop_nmi_interrupt	A
		r_cgc_oscillation_stop_interrupt	A
	r_cg_cgc.h	-	-
Voltage detection circuit (LVDA)	r_cg_lvd.c	R_LVDn_Create	A
		R_LVDn_Start	A
		R_LVDn_Stop	A
	r_cg_lvd_user.c	R_LVDn_Create_UserInit	M
		r_lvd_lvdn_interrupt	A
	r_cg_lvd.h	-	-
Clock frequency accuracy measurement circuit (CAC)	r_cg_cac.c	R_CAC_Create	A
		R_CAC_Start	A
		R_CAC_Stop	A
	r_cg_cac_user.c	R_CAC_Create_UserInit	M
		r_cac_mendf_interrupt	A
		r_cac_ferrf_interrupt	A
		r_cac_ovff_interrupt	A
	r_cg_cac.h	-	-
Low power consumption	r_cg_lpc.c	R_LPC_Create	A
		R_LPC_AllModuleClockStop	A
		R_LPC_ChangeSleepModeReturnClock	A
		R_LPC_Sleep	A
		R_LPC_DeepSleep	A
		R_LPC_DeepSoftwareStandby	A
		R_LPC_SoftwareStandby	A
		R_LPC_ChangeOperatingPowerControl	A
	r_cg_lpc_user.c	R_LPC_Create_UserInit	M
	r_cg_lpc.h	-	-

Table 2.3 Output File List (3/10)

Peripheral Function	File Name	API Function Name	output (*1)
Interrupt controller (ICU)	r_cg_icu.c	R_ICU_Create	A
		R_ICU_IRQn_Start	A
		R_ICU_IRQn_Stop	A
		R_ICU_Software_Start	A
		R_ICU_Software2_Start	A
		R_ICU_Software_Stop	A
		R_ICU_Software2_Stop	A
		R_ICU_SoftwareInterrupt_Generate	A
		R_ICU_SoftwareInterrupt2_Generate	A
	r_cg_icu_user.c	R_ICU_Create_UserInit	M
		r_icu_irqn_interrupt	A
		r_icu_software_interrupt	A
		r_icu_software2_interrupt	A
		r_icu_nmi_interrupt	A
	r_cg_icu.h	-	-
Buses	r_cg_bsc.c	R_BSC_Create	A
		R_BSC_Error_Monitoring_Start	A
		R_BSC_Error_Monitoring_Stop	A
		R_BSC_InitializeSDRAM	A
	r_cg_bsc_user.c	R_BSC_Create_UserInit	M
		r_bsc_buserr_interrupt	A
	r_cg_bsc.h	-	-
DMA Controller(DMAC)	r_cg_dmac.c	R_DMAC_Create	A
		R_DMACn_Start	A
		R_DMACn_Stop	A
		R_DMACn_Set_SoftwareTrigger	A
		R_DMACn_Clear_SoftwareTrigger	A
	r_cg_dmac_user.c	r_dmac_dmacni_interrupt	A
		r_dmacn_callback_transfer_end	A
		r_dmacn_callback_transfer_escape_end	A
		R_DMAC_Create_UserInit	M
	r_cg_dmac.h	-	-
Data transfer controller (DTC)	r_cg_dtc.c	R_DTC_Create	A
		R_DTCm_Start	A
		R_DTCm_Stop	A
	r_cg_dtc_user.c	R_DTC_Create_UserInit	M
	r_cg_dtc.h	-	-

Table 2.4 Output File List (4/10)

Peripheral Function	File Name	API Function Name	output (*1)
Event link controller (ELC)	r_cg_elc.c	R_ELC_Create	A
		R_ELC_Start	A
		R_ELC_Stop	A
		R_ELC_GenerateSoftwareEvent	A
		R_ELC_Set_PortBuffern	A
		R_ELC_Get_PortBuffern	A
	r_cg_elc_user.c	R_ELC_Create_UserInit	M
		r_elc_elsrni_interrupt	A
	r_cg_elc.h	-	-
I/O ports	r_cg_port.c	R_PORT_Create	A
	r_cg_port_user.c	R_PORT_Create_UserInit	M
	r_cg_port.h	-	-
Multi-function timer pulse unit 2 (MTU2)	r_cg_mtu2.c	R_MTU2_Create	A
		R_MTU2_Cn_Start	A
		R_MTU2_Cn_Stop	A
	r_cg_mtu2_user.c	R_MTU2_Create_UserInit	M
		r_mtu2_tgimn_interrupt	A
		r_mtu2_cj_tgimn_interrupt	A
		r_mtu2_tcivn_interrupt	A
		r_mtu2_cj_tcivn_interrupt	A
		r_mtu2_tciun_interrupt	A
	r_cg_mtu2.h	-	-
Multi-function timer pulse unit 3 (MTU3)	r_cg_mtu3.c	R_MTU3_Create	A
		R_MTU3_Cn_Start	A
		R_MTU3_Cn_Stop	A
	r_cg_mtu3_user.c	R_MTU3_Create_UserInit	M
		r_mtu3_tgimn_interrupt	A
		r_mtu3_cj_tgimn_interrupt	A
		r_mtu3_tcivn_interrupt	A
		r_mtu3_cj_tcivn_interrupt	A
		r_mtu3_tciun_interrupt	A
	r_cg_mtu3.h	-	-
Port output enable 2 (POE2)	r_cg_poe2.c	R_POE2_Create	A
		R_POE2_Start	A
		R_POE2_Stop	A
		R_POE2_Set_HiZ_MTUn	A
		R_POE2_Clear_HiZ_MTUn	A
	r_cg_poe2_user.c	R_POE2_Create_UserInit	M
		r_poe2_oein_interrupt	A
	r_cg_poe2.h	-	-

Table 2.5 Output File List (5/10)

Peripheral Function	File Name	API Function Name	output (*1)
Port output enable 3 (POE3)	r_cg_poe3.c	R_POE3_Create	A
		R_POE3_Start	A
		R_POE3_Stop	A
		R_POE3_Set_HiZ_MTUn	A
		R_POE3_Clear_HiZ_MTUn	A
		R_POE3_Set_HiZ_GPTn	A
		R_POE3_Clear_HiZ_GPTn	A
	r_cg_poe3_user.c	R_POE3_Create_UserInit	M
		r_poe3_oein_interrupt	A
	r_cg_poe3.h	-	-
General PWM timer (GPT)	r_cg_gpt.c	R_GPT_Create	A
		R_GPTn_Start	A
		R_GPTn_Stop	A
		R_GPTn_HardwareStart	A
		R_GPTn_HardwareStop	A
	r_cg_gpt_user.c	R_GPT_Create_UserInit	M
		r_gpt_gtcimn_interrupt	A
		r_gpt_gtcivn_interrupt	A
		r_gpt_gtciun_interrupt	A
		r_gpt_gdten_interrupt	A
		r_gpt_etgip_interrupt	A
		r_gpt_etgin_interrupt	A
	r_cg_gpt.h	-	-
16-bit timer pulse unit (TPU)	r_cg_tpu.c	R_TPU_Create	A
		R_TPUn_Start	A
		R_TPUn_Stop	A
	r_cg_tpu_user.c	R_TPU_Create_UserInit	M
		r_tpu_tginm_interrupt	A
		r_tpu_tcinv_interrupt	A
		r_tpu_tcinu_interrupt	A
	r_cg_tpu.h	-	-
8-bit timer (TMR)	r_cg_tmr.c	R_TMR_Create	A
		R_TMRn_Start	A
		R_TMRn_Stop	A
	r_cg_tmr_user.c	R_TMR_Create_UserInit	M
		r_tmr_cmimn_interrupt	A
		r_tmr_ovin_interrupt	A
	r_cg_tmr.h	-	-
Programmable pulse generator (PPG)	r_cg_ppg.c	R_PPG_Create	A
	r_cg_ppg_user.c	R_PPG_Create_UserInit	M
	r_cg_ppg.h	-	-

Table 2.6 Output File List (6/10)

Peripheral Function	File Name	API Function Name	output (*1)
Compare match timer (CMT)	r_cg_cmt.c	R_CMTn_Create	A
		R_CMTn_Start	A
		R_CMTn_Stop	A
	r_cg_cmt_user.c	R_CMTn_Create_UserInit	M
		r_cmt_cmin_interrupt	A
	r_cg_cmt.h	-	-
Compare match timer W (CMTW)	r_cg_cmtw.c	R_CMTWn_Create	A
		R_CMTWn_Start	A
		R_CMTWn_Stop	A
	r_cg_cmtw_user.c	R_CMTWn_Create_UserInit	M
		r_cmtw_cmwin_interrupt	A
		r_cmtw_icmin_interrupt	A
		r_cmtw_ocmin_interrupt	A
	r_cg_cmtw.h	-	-
Realtime clock (RTC)	r_cg_rtc.c	R_RTC_Create	A
		R_RTC_Set_CalendarAlarm	A
		R_RTC_Set_BinaryAlarm	A
		R_RTC_Set_ConstPeriodInterruptOn	A
		R_RTC_Set_ConstPeriodInterruptOff	A
		R_RTC_Set_CarryInterruptOn	A
		R_RTC_Set_CarryInterruptOff	A
		R_RTC_Set_RTCOUTOn	A
		R_RTC_Set_RTCOUTOff	A
		R_RTC_Start	A
		R_RTC_Stop	A
		R_RTC_Restart	A
		R_RTC_Set_CalendarCounterValue	A
		R_RTC_Get_CalendarCounterValue	A
		R_RTC_Set_BinaryCounterValue	A
		R_RTC_Get_BinaryCounterValue	A
		R_RTC_Get_CalendarTimeCaptureValuen	A
		R_RTC_Get_BinaryTimeCaptureValuen	A
	r_cg_rtc_user.c	R_RTC_Create_UserInit	M
		r_rtc_alm_interrupt	A
		r_rtc_prd_interrupt	A
		r_rtc_cup_interrupt	A
	r_cg_rtc.h	-	-

Table 2.7 Output File List (7/10)

Peripheral Function	File Name	API Function Name	output (*1)
Watchdog timer (WDT)	r_cg_wdt.c	R_WDT_Create	A
		R_WDT_Restart	A
	r_cg_wdt_user.c	R_WDT_Create_UserInit	M
		r_wdt_nmi_interrupt	A
		r_wdt_wuni_interrupt	A
	r_cg_wdt.h	-	-
Independent watchdog timer (IWDT)	r_cg_iwdt.c	R_IWDT_Create	A
		R_IWDT_Restart	A
	r_cg_iwdt_user.c	R_IWDT_Create_UserInit	M
		r_iwdt_nmi_interrupt	A
		r_iwdt_iwuni_interrupt	A
	r_cg_iwdt.h	-	-
Serial communications interface (SCI)	r_cg_sci.c	R_SCIn_Create	A
		R_SCIn_Start	A
		R_SCIn_Stop	A
		R_SCIn_Serial_Send	A
		R_SCIn_Serial_Receive	A
		R_SCIn_Serial_Multiprocessor_Send	A
		R_SCIn_Serial_Multiprocessor_Receive	A
		R_SCIn_Serial_Send_Receive	A
		R_SCIn_SmartCard_Send	A
		R_SCIn_SmartCard_Receive	A
		R_SCIn_IIC_Master_Send	A
		R_SCIn_IIC_Master_Receive	A
		R_SCIn_SPI_Master_Send	A
		R_SCIn_SPI_Master_Send_Receive	A
		R_SCIn_SPI_Slave_Send	A
		R_SCIn_SPI_Slave_Send_Receive	A
		R_SCIn_IIC_StartCondition	A
		R_SCIn_IIC_StopCondition	A
	r_cg_sci_user.c	R_SCIn_Create_UserInit	M
		r_scin_transmitend_interrupt	A
		r_scin_transmit_interrupt	A
		r_scin_receive_interrupt	A
		r_scin_receiveerror_interrupt	A
		r_scin_callback_transmitend	A
		r_scin_callback_receiveend	A
		r_scin_callback_receiveerror	A
	r_cg_sci.h	-	-

Table 2.8 Output File List (8/10)

Peripheral Function	File Name	API Function Name	output (*1)
FIFO embedded serial communications interface (SCIFA)	r_cg_scifa.c	R_SCIFAn_Create	A
		R_SCIFAn_Start	A
		R_SCIFAn_Stop	A
		R_SCIFAn_Serial_Send	A
		R_SCIFAn_Serial_Receive	A
		R_SCIFAn_Serial_Send_Receive	A
	r_cg_scifa_user.c	R_SCIFAn_Create_UserInit	M
		r_scifan_teif_interrupt	A
		r_scifan_txif_interrupt	A
		r_scifan_rxif_interrupt	A
		r_scifan_erif_interrupt	A
		r_scifan_brif_interrupt	A
		r_scifan_drif_interrupt	A
		r_scifan_callback_transmitend	A
		r_scifan_callback_receiveend	A
		r_scifan_callback_error	A
	r_cg_scifa.h	-	-
I2C bus interface (RIIC)	r_cg_riic.c	R_RIICn_Create	A
		R_RIICn_Start	A
		R_RIICn_Stop	A
		R_RIICn_Master_Send	A
		R_RIICn_Master_Receive	A
		R_RIICn_Slave_Send	A
		R_RIICn_Slave_Receive	A
		R_RIICn_StartCondition	A
		R_RIICn_StopCondition	A
	r_cg_riic_user.c	R_RIICn_Create_UserInit	M
		r_riicn_error_interrupt	A
		r_riicn_receive_interrupt	A
		r_riicn_transmit_interrupt	A
		r_riicn_transmitend_interrupt	A
		r_riicn_callback_receiveerror	A
		r_riicn_callback_transmitend	A
		r_riicn_callback_receiveend	A
	r_cg_riic.h	-	-

Table 2.9 Output File List (9/10)

Peripheral Function	File Name	API Function Name	output (*1)
Serial peripheral interface (RSPI)	r_cg_rspi.c	R_RSPIn_Create	A
		R_RSPIn_Start	A
		R_RSPIn_Stop	A
		R_RSPIn_Send	A
		R_RSPIn_Send_Receive	A
	r_cg_rspi_user.c	R_RSPIn_Create_UserInit	M
		r_rspin_receive_interrupt	A
		r_rspin_transmit_interrupt	A
		r_rspin_error_interrupt	A
		r_rspin_idle_interrupt	A
		r_rspin_callback_receiveend	A
		r_rspin_callback_error	A
		r_rspin_callback_transmitend	A
	r_cg_rspi.h	-	-
CRC calculator (CRC)	r_cg_crc.c	R_CRC_SetCRC8	A
		R_CRC_SetCRC16	A
		R_CRC_SetCCITT	A
		R_CRC_SetCRC32	A
		R_CRC_SetCRC32C	A
		R_CRC_Input_Data	A
		R_CRC_Get_Result	A
	r_cg_crc.h	-	-
12-bit A/D converter (S12AD)	r_cg_s12ad.c	R_S12ADn_Create	A
		R_S12ADn_Start	A
		R_S12ADn_Stop	A
		R_S12ADn_Get_ValueResult	A
		R_S12ADn_Set_CompareValue	A
	r_cg_s12ad_user.c	R_S12ADn_Create_UserInit	M
		r_s12adn_interrupt	A
		r_s12adn_groupb_interrupt	A
		r_s12adn_compare_interrupt	A
	r_cg_s12ad.h	-	-
D/A converter (DA)	r_cg_da.c	R_DA_Create	A
		R_DAm_Start	A
		R_DAm_Stop	A
		R_DAm_Set_ConversionValue	A
	r_cg_da_user.c	R_DA_Create_UserInit	M
	r_cg_da.h	-	-

Table 2.10 Output File List (10/10)

Peripheral Function	File Name	API Function Name	output (*1)
12-bit converter (R12DA)	r_cg_r12da.c	R_R12DA_Create	A
		R_R12DAn_Start	A
		R_R12DAn_Stop	A
		R_R12DAn_Set_ConversionValue	A
		R_R12DA_Sync_Start	A
		R_R12DA_Sync_Stop	A
	r_cg_r12da_user.c	R_R12DA_Create_UserInit	M
	r_cg_r12da.h	-	-
Comparator B (CMPB)	r_cg_cmpb.c	R_CMPB_Create	A
		R_CMPBn_Start	A
		R_CMPBn_Stop	A
	r_cg_cmpb_user.c	R_CMPB_Create_UserInit	M
		r_cmpb_cmpbn_interrupt	A
	r_cg_cmpb.h	-	-
Data operation circuit (DOC)	r_cg_doc.c	R_DOC_Create	A
		R_DOC_SetMode	A
		R_DOC_WriteData	A
		R_DOC_GetResult	A
		R_DOC_ClearFlag	A
	r_cg_doc_user.c	R_DOC_Create_UserInit	M
		r_doc_dopcf_interrupt	A
	r_cg_doc.h	-	-
Low power timer (LPT)	r_cg_lpt.c	R_LPT_Create	A
		R_LPT_Start	A
		R_LPT_Stop	A
	r_cg_lpt_user.c	R_LPT_Create_UserInit	M
	r_cg_lpt.h	-	-
Comparator C (CMPC)	r_cg_cmpc.c	R_CMPC_Create	A
		R_CMPCn_Start	A
		R_CMPCn_Stop	A
	r_cg_cmpc_user.c	R_CMPC_Create_UserInit	M
		r_cmpc_cmpcn_interrupt	A
	r_cg_cmpc.h	-	-
LCD controller / driver (LCD)	r_cg_lcd.c	R_LCD_Create	A
		R_LCD_Start	A
		R_LCD_Stop	A
		R_LCD_Voltage_On	A
		R_LCD_Voltage_Off	A
	r_cg_lcd_user.c	R_LCD_Create_UserInit	M
	r_cg_lcd.h	-	-

*1 In case of [API output control] setting are default ([Output all API functions according to the setting]).
A : Output by settings on each peripheral functions panel automatically.
M : Output by the file used setting in API property.

3. API FUNCTIONS

This appendix describes the API functions output by the Code Generator.

3.1 Overview

Below are the naming conventions for API functions output by the Code Generator.

- Macro names are in ALL CAPS.
  The number in front of the macro name is a hexadecimal value; this is the same value as the macro value.

- Local variable names are in all lower case.

- Global variable names start with a "g" and use Camel Case.

- Names of pointers to global variables start with a "gp" and use Camel Case.

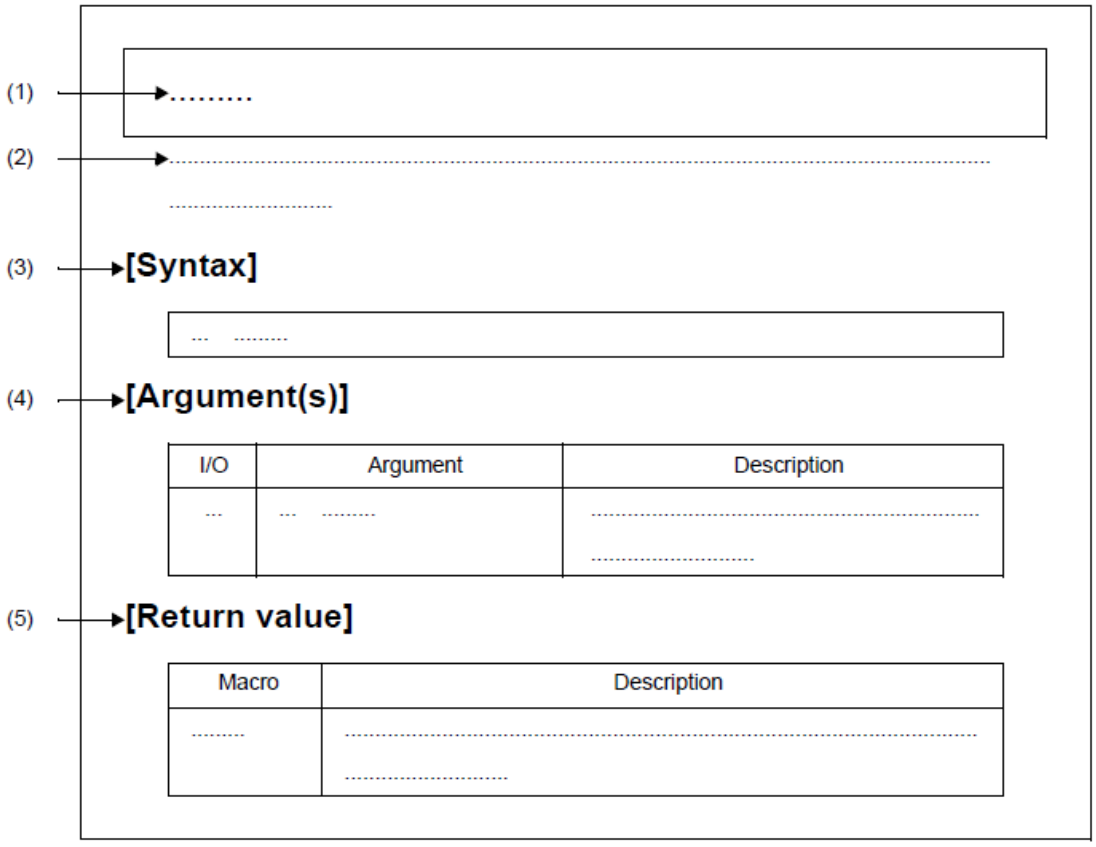
- Names of elements in enum statements are in ALL CAPS.

Remarks In the generated code by the code generator tool, the for statement, the while statement, the do-while statement (loop processing) are used in register setting reflected waiting process etc. If fail-safe processing for infinite loop is required, check the generated code and add processing.

3.2 Function Reference

This section describes the API functions output by the Code Generator, using the following notation format.

Figure 3.1 Notation Format of API Functions



(1) Name
Indicates the name of the API function.

(2) Outline
Outlines the functions of the API function

(3) [Syntax]
Indicates the format to be used when describing an API function to be called in C language.

(4) [Argument(s)]
API function arguments are explained in the following format.

I/O	Argument	Descripton
(a)	(b)	(c)

(a) I/O
Argument classification
I … Input argument
O … Output argument

(b) Argument
Argument data type

(c) Description
Description of argument

(5) [Return value]
API function return value is explained in the following format.

Macro	Description
(a)	(b)

(a) Macro
Macro of return value

(b) Description
Description of return value

3.2.1 Common

Below is a list of API functions output by the Code Generator for common use.

Performs processing in response to the exception (other than undefined instruction exception, reset, non-maskable interrupt and unconditional trap).

Table 3.1 API Functions: [Common]

API Function Name	Function
r_undefined_exception	Performs processing in response to the undefined instruction exception.
PowerON_Reset	Performs processing in response to the reset.
PowerON_Reset_PC	Performs processing in response to the reset.
r_privileged_exception	Performs processing in response to the privileged instruction exception.
r_floatingpoint_exception	Performs processing in response to the floating-point exception.
r_access_exception	Performs processing in response to the access exception.
r_nmi_exception	Performs processing in response to the non-maskable interrupt.
r_brk_exception	Performs processing in response to the unconditional trap.
r_reserved_exception	Performs processing in response to the exception (other than undefined instruction exception, reset, non-maskable interrupt and unconditional trap).
HardwareSetup	Performs initialization necessary to control the various hardwares.
R_Systeminit	Performs initialization necessary to control the various peripheral functions.
main	This is a main function.
R_MAIN_UserInit	Performs user-defined initialization.
r_icu_group_n_interrupt	Performs processing in response to the group interrupt.
_NMI_handler	Performs processing in response to the non-maskable interrupt.
_low_level_init	Performs initialization necessary to control the various hardwares.

r_undefined_exception

Performs processing in response to the undefined instruction exception.

Remark This API function is called to run interrupt processing in response to an undefined instruction exception occurred when detecting the undefined instruction (unimplemented instruction) execution.

[Syntax]

```
void    r_undefined_exception ( void );
```

[Argument(s)]

None.

[Return value]

None.

PowerON_Reset

Performs processing in response to the reset.

Remark This API function is called to run interrupt processing for an internal reset by the power-on reset circuit.

[Syntax]

```
void    PowerON_Reset ( void );
```

[Argument(s)]

None.

[Return value]

None.

PowerON_Reset_PC

Performs processing in response to the reset.

Remark This API function is called to run interrupt processing for an internal reset by the power-on reset circuit.

[Syntax]

```
void    PowerON_Reset_PC ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_privileged_exception

Performs processing in response to the privileged instruction exception.

Remark This API function is called to run interrupt processing in response to a privilegedundefined instruction exception occurred when detecting the execution of a privileged instruction in user mode.

[Syntax]

```
void      r_privileged_exception ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_floatingpoint_exception

Performs processing in response to the floating-point exception.

Remark This API function is called to run interrupt processing in response to a floating-point exception occurred when detecting the five exception events (overflow, underflow, inexact, division-by-zero, and invalid operation) specified in the IEEE754 standard and another floating-point exception that is generated on detection of unimplemented processing.

[Syntax]

```
void    r_floatingpoint_exception ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_access_exception

Performs processing in response to the access exception.

Remark This API function is called to run interrupt processing in response to an access exception occurred when detecting the memory-protection error, and data memory protection error.

[Syntax]

```
void    r_access_exception ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_nmi_exception

Performs processing in response to the non-maskable interrupt.

Remark This API function is called to run interrupt processing for the non-maskable interrupt.

[Syntax]

```
void    r_nmi_exception ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_brk_exception

Performs processing in response to the unconditional trap.

Remark This API function is called to run interrupt processing for an unconditional trap.

[Syntax]

```
void    r_brk_exception ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_reserved_exception

Performs processing in response to the exception (other than undefined instruction exception, reset, non-maskable interrupt and unconditional trap).

Remark This API function is called to run interrupt processing for the exceptions other than undefined instruction exception, reset, non-maskable interrupt, and unconditional trap.

[Syntax]

```
void    r_reserved_exception ( void );
```

[Argument(s)]

None.

[Return value]

None.

HardwareSetup

Performs initialization necessary to control the various hardwares.

Remark This API function is called as the PowerON_Reset callback routine.

[Syntax]

```
void    HardwareSetup ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_Systeminit

Performs initialization necessary to control the various peripheral functions.

Remark This API function is called as the HardwareSetup callback routine.

[Syntax]

```
void    R_Systeminit ( void );
```

[Argument(s)]

None.

[Return value]

None.

main

This is a main function.

Remark This API function is called as the PowerON_Reset callback routine.

[Syntax]

```
void    main ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_MAIN_UserserInit

Performs user-defined initialization.

Remark This API function is called as the main callback routine.

[Syntax]

```
void    R_MAIN_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_icu_group_*n*_interrupt

Performs processing in response to the group interrupts.

[Syntax]

```
void    r_icu_group_n_interrupt ( void );
```

Remark *n* is the group interrupt number.

[Argument(s)]

None.

[Return value]

None.

_NMI_handler

Performs processing in response to the non-maskable interrupt.

Remark This API function is called to run interrupt processing for the non-maskable interrupt.

[Syntax]

```
void    _NMI_handler ( void );
```

[Argument(s)]

None.

[Return value]

None.

_low_level_init

Performs initialization necessary to control the various hardwares.

[Syntax]

```
void    _low_level_init ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.2 Clock generation circuit

Below is a list of API functions output by the Code Generator for clock generation circuit use.

Table 3.2 API Functions: [Clock Generation Circuit]

API Function Name	Function
R_CGC_Create	Performs initialization required to control the clock generation circuit.
R_CGC_Create_UserInit	Performs user-defined initialization relating to the clock generation circuit.
r_cgc_oscillation_stop_interrupt	Performs processing in response to the osillation stop detection interrupt.
r_cgc_oscillation_stop_nmi_interrupt	Performs processing in response to the oscillation stop detection NMI.
R_CGC_Set_ClockMode	Sets the clock source.

R_CGC_Create

Performs initialization required to control the clock generation circuit.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_CGC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CGC_Create_UserInit

Performs user-defined initialization relating to the clock generation circuit.

Remark This API function is called as the R_CGC_Create callback routine.

[Syntax]

```
void    R_CGC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_cgc_oscillation_stop_interrupt

Performs processing in response to the oscillation stop detection interrupt.

Remark This API function is called to run interrupt processing for the oscillation stop detection interrupt, which is generated when the clock generation circuit detects oscillation by the main clock having stopped.

[Syntax]

```
static void     r_cgc_oscillation_stop_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_cgc_oscillation_stop_nmi_interrupt

Performs processing in response to the oscillation stop detection NMI.

[Syntax]

```
static void      r_cgc_oscillation_stop_nmi_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CGC_Set_ClockMode

Sets the clock source.

[Syntax]

```
#include        "r_cg_macrodriver.h"
#include        "r_cg_cgc.h"
MD_STATUS     R_CGC_Set_ClockMode ( clock_mode_t mode );
```

[Argument(s)]

I/O	Argument	Description
I	clock_mode_t *mode*;	Clock source type MAINCLK : Main clock oscillator SUBCLK : Sub-clock oscillator PLLCLK : PLL circuit HOCO : High-speed on-chip oscillator LOCO : Low-speed on-chip oscillator

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Exit with error (abend)
MD_ARGERROR	Invalid argument *mode* specification

Usage example

When stop of the main clock oscillator is detected, the clock source is switched to the high-speed on-chip oscillator clock.

[GUI setting example]

Clock Generator				
	CGC			Used
			Main clock oscillator forced oscillation	Unused
			Main clock oscillation source	Resonator
			Main clock oscillation source Frequency	24(MHz)
			Oscillator wait time	11000(μs)
			Oscillation stop detection function	Enabled (with interrupt request output)
			OSTDI Priority	Level 15 (highest)
			PLL Operation	Unused
			SubCLK Operation	Unused
			HOCO Operation	Used
			HOCO Frequency	16 (MHz)
			LOCO Operation	Used
			LOCO Frequency	240 (kHz)
			IWDT Operation	Unused
			Clock source	Main clock oscillator
			System clock (ICLK)	x 1/4 6 (MHz)
			Peripheral module clock (PCLKA)	x 1/4 6 (MHz)
			Peripheral module clock (PCLKB)	x 1/4 6 (MHz)
			Peripheral module clock (PCLKC)	x 1/4 6 (MHz)
			Peripheral module clock (PCLKD)	x 1/4 6 (MHz)
			External bus clock (BCLK)	x 1/4 6 (MHz)
			Flash IF clock (FCLK)	x 1/4 6 (MHz)
			USB clock (UCLK)	x 1/3 8 (MHz)
			RTC clock setting	Unused
			BCLK Operation	Unused
			SDCLK Operation	Unused
			Debug interface setting	Unused

[API setting example]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_cgc_user.c

```
static void r_cgc_oscillation_stop_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Change clock generator operation mode */
    R_CGC_Set_ClockMode(HOCO);
    /* End user code. Do not edit comment generated here */
}
```

3.2.3 Voltage detection circuit (LVDA)

Below is a list of API functions output by the Code Generator for voltage detection circuit use.

Table 3.3 API Functions: [Voltage Detection Circuit]

API Function Name	Function
R_LVDn_Create	Performs initialization necessary to control the voltage detection circuit.
R_LVDn_Create_UserInit	Performs user-defined initialization relating to the voltage detection circuit.
r_lvd_lvdn_interrupt	Performs processing in response to the voltage monitoring *n* interrupt.
R_LVDn_Start	Starts voltage monitoring (when in interrupt mode, and interrupt & reset mode).
R_LVDn_Stop	Ends voltage monitoring (when in interrupt mode, and interrupt & reset mode).

R_LVD*n*_Create

Performs initialization necessary to control the voltage detection circuit.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_LVDn_Create ( void );
```

Remark *n* is the circuit number.

[Argument(s)]

None.

[Return value]

None.

R_LVD*n*_Create_UserInit

Performs user-defined initialization relating to the voltage detection circuit.

Remark This API function is called as the R_LVDn_Create callback routine.

[Syntax]

```
void R_LVDn_Create_UserInit ( void );
```

Remark *n* is the circuit number.

[Argument(s)]

None.

[Return value]

None.

r_lvd_lvd*n*_interrupt

Performs processing in response to the voltage monitoring *n* interrupt.

Remark This API function is called to run interrupt processing for the voltage monitoring *n* interrupt, which is generated when the voltage detection circuit detects the voltage being dropped.

[Syntax]

```
static void    r_lvd_lvdn_interrupt ( void );
```

Remark *n* is the circuit number.

[Argument(s)]

None.

[Return value]

None.

R_LVD*n*_Start

Starts voltage monitoring (when in interrupt mode, and interrupt & reset mode).

[Syntax]

```
void    R_LVDn_Start ( void );
```

Remark *n* is the circuit number.

[Argument(s)]

None.

[Return value]

None.

R_LVD*n*_Stop

Ends voltage monitoring (when in interrupt mode, and interrupt & reset mode).

[Syntax]

```
void    R_LVDn_Stop ( void );
```

Remark *n* is the circuit number.

[Argument(s)]

None.

[Return value]

None.

Usage example

Perform software reset at voltage monitoring interrupt.

[GUI setting example]

Voltage Detection Circuit				Used
	LVD1			Used
			Voltage detection 1 circuit operation setting	Used
			Enable voltage monitoring digital filter	Unused
			Voltage detection level	2.85(V)
			Voltage monitoring circuit mode	Voltage monitoring 1 interrupt enabled when Vdet1 is crossed
			Interrupt setting	Maskable interrupt
			Interrupt ELC event generation condition	When VCC < Vdet1 (drop) is detected
			Priority	Level 15 (highest)

[API setting example]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start the LVD1 operation */
    R_LVD1_Start();

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_lvd_user.c

```
static void r_lvd_lvd1_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Software reset */
    SYSTEM.PRCR.WORD = 0xA502;
    SYSTEM.SWRR = 0xA501;
    /* End user code. Do not edit comment generated here */
}
```

3.2.4 Clock frequency accuracy measurement circuit (CAC)

Below is a list of API functions output by the Code Generator for clock frequency accuracy measurement circuit use.

Table 3.4 API Functions: [Clock Frequency Accuracy Measurement Circuit]

API Function Name	Function
R_CAC_Create	Performs initialization necessary to control the clock frequency accuracy measurement circuit.
R_CAC_Create_UserInit	Performs user-defined initialization relating to the clock frequency accuracy measurement circuit.
r_cac_mendf_interrupt	Performs processing in response to the measurement end interrupt.
r_cac_ferrf_interrupt	Performs processing in response to the frequency error interrupt.
r_cac_ovff_interrupt	Performs processing in response to the overflow interrupt.
R_CAC_Start	Starts measurement of the accuracy of the clock frequency.
R_CAC_Stop	Ends measurement of the accuracy of the clock frequency.

R_CAC_Create

Performs initialization necessary to control the clock frequency accuracy measurement circuit.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_CAC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CAC_Create_UserInit

Performs user-defined initialization relating to the clock frequency accuracy measurement circuit.

Remark This API function is called as the R_CAC_Create callback routine.

[Syntax]

```
void    R_CAC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_cac_mendf_interrupt

Performs processing in response to the measurement end interrupt.

Remark This API function is called to run interrupt processing for the measurement end interrupt, which is generated when the clock frequency accuracy measurement circuit detects the valid edge of the reference signal.

[Syntax]

```
static void     r_cac_mendf_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_cac_ferrf_interrupt

Performs processing in response to the frequency error interrupt.

Remark This API function is called to run interrupt processing for the frequency error interrupt, which is generated when the clock frequency is not in the allowed range (from the minimum to the maximum value).

[Syntax]

```
static void     r_cac_ferrf_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_cac_ovff_interrupt

Performs processing in response to the overflow interrupt.

Remark This API function is called to run interrupt processing for the overflow interrupt, which is generated when the counter overflows.

[Syntax]

```
static void     r_cac_ovff_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CAC_Start

Starts measurement of the accuracy of the clock frequency.

[Syntax]

```
void    R_CAC_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CAC_Stop

Ends measurement of the accuracy of the clock frequency.

[Syntax]

```
void    R_CAC_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

Usage example

Count frequency error.

[GUI setting example]

Clock Frequency Accuracy Measurement Circuit				Used
	CAC			Used
			CAC operation setting	Used
			Measurement reference setting Clock select	Main clock
			Measurement reference setting Frequency	x 1/32 (0.75)(MHz)
			Digital filter selection	Disabled
			Valid edge	Rising
			Measurement target setting Clock select	Main clock
			Measurement target setting Frequency	x 1 (24)(MHz)
			Maximum positive deviation	5(%)(Actual value：3.125)
			Maximum negative deviation	5(%)(Actual value：6.25)
			Enable frequency error interrupt (FERRF)	Used
			Priority (Group BL0)(FERRF)	Level 15 (highest)
			Enable measurement end interrupt (MENDF)	Used
			Priority (Group BL0)(MENDF)	Level 15 (highest)
			Enable overflow interrupt (OVFF)	Used
			Priority (Group BL0)(OVFF)	Level 15 (highest)

Interrupt Controller Unit				Used
	ICU			Used
		Group		Used
			Group BL0	Used
			Group BL0 Priority	Level 15 (highest)

[API setting example]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Enable clock frequency measurement */
    R_CAC_Start();

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_cac_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_cac_ferri_cnt;
/* End user code. Do not edit comment generated here */

void R_CAC_Create_UserInit(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Reset the error countor */
    g_cac_ferri_cnt = 0U;
    /* End user code. Do not edit comment generated here */
}

void r_cac_ferrf_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Add the error countor */
    g_cac_ferri_cnt++;
    /* End user code. Do not edit comment generated here */
}
```

3.2.5 Low power consumption

Below is a list of API functions output by the Code Generator for low power consumption use.

Table 3.5 API Functions: [Low Power Consumption]

API Function Name	Function
R_LPC_Create	Performs initialization required to control the low power consumption.
R_LPC_Create_UserInit	Performs user-defined initialization relating to the low power consumption.
R_LPC_AllModuleClockStop	Stops the clock for all modules.
R_LPC_ChangeSleepModeReturnClock	Sets the clock source that is selected following release from sleep mode.
R_LPC_Sleep	Transits the low power consumption mode of the MCU to the sleep mode.
R_LPC_DeepSleep	Transits the low power consumption mode of the MCU to the deep sleep mode.
R_LPC_DeepSoftwareStandby	Transits the low power consumption mode of the MCU to the deep software standby mode.
R_LPC_SoftwareStandby	Transits the low power consumption mode of the MCU to the software standby mode.
R_LPC_ChangeOperatingPowerControl	Changes the operating power control mode of the MCU.

R_LPC_Create

Performs initialization required to control the low power consumption.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_LPC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LPC_Create_UserInit

Performs user-defined initialization relating to the low power consumption.

Remark This API function is called as the R_LPC_Create callback routine.

[Syntax]

```
void    R_LPC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LPC_AllModuleClockStop

Stops the clock for all modules.

[Syntax]

```
#include          "r_cg_macrodriver.h"
MD_STATUS     R_LPC_AllModuleClockStop ( void );
```

[Argument(s)]

None.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Exit with error (abend)

R_LPC_ChangeSleepModeReturnClock

Sets the clock source that is selected following release from sleep mode.

[Syntax]

```
#include        "r_cg_macrodriver.h"
#include        "r_cg_lpc.h"
MD_STATUS     R_LPC_ChangeSleepModeReturnClock ( return_clock_t clock );
```

[Argument(s)]

I/O	Argument	Description
I	return_clock_t *clock*;	Clock source type RETURN_LOCO : Low-speed on-chip oscillator RETURN_HOCO : High-speed on-chip oscillator RETURN_MAIN_CLOCK : Main clock oscillator RETURN_DISABLE : Switching of the clock source does not proceed.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Change to the low-speed operating mode ended abnormally.
MD_ARGERROR	Invalid argument *clock* specification

R_LPC_Sleep

Transits the low power consumption mode of the MCU to the sleep mode.

[Syntax]

```
#include        "r_cg_macrodriver.h"
MD_STATUS     R_LPC_Sleep ( void );
```

[Argument(s)]

None.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Exit with error (abend)

R_LPC_DeepSleep

Transits the low power consumption mode of the MCU to the deep sleep mode.

[Syntax]

```
#include          "r_cg_macrodriver.h"
MD_STATUS     R_LPC_DeepSleep ( void );
```

[Argument(s)]

None.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Exit with error (abend)

R_LPC_DeepSoftwareStandby

Transits the low power consumption mode of the MCU to the deep software standby mode.

[Syntax]

```
#include          "r_cg_macrodriver.h"
MD_STATUS     R_LPC_DeepSoftwareStandby ( void );
```

[Argument(s)]

None.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Exit with error (abend)

R_LPC_SoftwareStandby

Transits the low power consumption mode of the MCU to the software standby mode.

[Syntax]

```
#include          "r_cg_macrodriver.h"
MD_STATUS     R_LPC_SoftwareStandby ( void );
```

[Argument(s)]

None.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Exit with error (abend)

R_LPC_ChangeOperatingPowerControl

Changes the operating power control mode of the MCU.

[Syntax]

```
#include        "r_cg_macrodriver.h"
#include        "r_cg_lpc.h"
MD_STATUS     R_LPC_ChangeOperatingPowerControl ( operating_mode_t mode );
```

[Argument(s)]

I/O	Argument	Description
I	operating_mode_t *mode*;	Operating power control mode type HIGH_SPEED : High-speed operating mode MIDDLE_SPEED : Middle-speed operating mode LOW_SPEED : Low-speed operating mode

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Change to the low-speed operating mode ended abnormally.
MD_ERROR2	Change to the middle-speed operating mode is ended abnormally.
MD_ARGERROR	Invalid argument *mode* specification

Usage example

Please refer to 'Usage example in 3.2.6 Interrupt controller (ICU)'.

3.2.6 Interrupt controller (ICU)

Below is a list of API functions output by the Code Generator for interrupt controller use.

Table 3.6 API Functions: [Interrupt Controller]

API Function Name	Function
R_ICU_Create	Performs initialization necessary to control the interrupt controller.
R_ICU_Create_UserInit	Performs user-defined initialization relating to the interrupt controller.
r_icu_irqn_interrupt	Performs processing in response to the external pin interrupts.
r_icu_software_interrupt	Performs processing in response to the software interrupt.
r_icu_software2_interrupt	Performs processing in response to the software interrupt2.
r_icu_nmi_interrupt	Performs processing in response to the NMI pin interrupt.
R_ICU_IRQn_Start	Allows detection of the external pin interrupt.
R_ICU_IRQn_Stop	Prohibits detection of the external pin interrupt.
R_ICU_Software_Start	Allows detection of the software interrupt.
R_ICU_Software2_Start	Allows detection of the software interrupt2.
R_ICU_Software_Stop	Prohibits detection of the software interrupt.
R_ICU_Software2_Stop	Prohibits detection of the software interrupt2.
R_ICU_SoftwareInterrupt_Generate	Generates the software interrupt.
R_ICU_SoftwareInterrupt2_Generate	Generates the software interrupt2.

R_ICU_Create

Performs initialization necessary to control the interrupt controller.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_ICU_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ICU_Create_UserInit

Performs user-defined initialization relating to the interrupt controller.

Remark This API function is called as the R_ICU_Create callback routine.

[Syntax]

```
void    R_ICU_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_icu_irq*n*_interrupt

Performs processing in response to the external pin interrupts.

Remark This API function is called to run interrupt processing for the external pin interrupts.

[Syntax]

```
static void     r_icu_irqn_interrupt ( void );
```

Remark *n* is the source number.

[Argument(s)]

None.

[Return value]

None.

r_icu_software_interrupt

Performs processing in response to the software interrupt.

Remark This API function is called to run the interrupt processing for the software interrupt, which is generated in response to calling of R_ICU_SoftwareInterrupt_Generate .

[Syntax]

```
static void     r_icu_software_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_icu_software2_interrupt

Performs processing in response to the software interrupt2.

Remark This API function is called to run the interrupt processing for the software interrupt, which is generated in response to calling of R_ICU_SoftwareInterrupt2_Generate .

[Syntax]

```
static void     r_icu_software2_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_icu_nmi_interrupt

Performs processing in response to the NMI pin interrupt.

Remark This API function is called to run interrupt processing for the NMI pin interrupt.

[Syntax]

```
static void     r_icu_nmi_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ICU_IRQ*n*_Start

Allows detection of the external pin interrupt.

[Syntax]

```
void    R_ICU_IRQn_Start ( void );
```

Remark *n* is the source number.

[Argument(s)]

None.

[Return value]

None.

R_ICU_IRQ*n*_Stop

Prohibits detection of the external pin interrupt.

[Syntax]

```
void    R_ICU_IRQn_Stop ( void );
```

Remark *n* is the source number.

[Argument(s)]

None.

[Return value]

None.

R_ICU_Software_Start

Allows detection of the software interrupt.

[Syntax]

```
void    R_ICU_Software_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ICU_Software2_Start

Allows detection of the software interrupt 2.

[Syntax]

```
void    R_ICU_Software2_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ICU_Software_Stop

Prohibits detection of the software interrupt.

[Syntax]

```
void    R_ICU_Software_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ICU_Software2_Stop

Prohibits detection of the software interrupt 2.

[Syntax]

```
void    R_ICU_Software2_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ICU_SoftwareInterrupt_Generate

Generates the software interrupt.

Remark r_icu_software_interrupt is called in response to calling od this API function.

[Syntax]

```
void    R_ICU_SoftwareInterrupt_Generate ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ICU_SoftwareInterrupt2_Generate

Generates the software interrupt 2.

Remark r_icu_software2_interrupt is called in response to calling od this API function.

[Syntax]

```
void    R_ICU_SoftwareInterrupt2_Generate ( void );
```

[Argument(s)]

None.

[Return value]

None.

Usage example

Return from sleep mode by external interrupt.

[GUI setting example]

Interrupt Controller Unit				Used
	ICU			Used
		IRQ5		Used
			IRQ5	Used
			Pin	P15
			Digital filter	No filter 0(MHz)
			Valid edge	Low level
			Priority	Level 15 (highest)

Low Power Consumption				Used
	LPC			Used
			LPC operation setting	Used
			Initial operating power control mode	High-speed operating mode
			Output of the address bus and bus control signals	Retain the output state in software standby mode or deep software standby mode
			Power supply to the on-chip RAM (000A4000h to 000A5FFFh) and USB resumption detecting unit	Power is not supplied in deep software standby mode
			Stop LVD and enable the low power consumption function of the power-on reset circuit	Unused
			Initial sleep mode return clock source	Disabled
			Enable request for release by IRQ0-DS (P30)	Unused
			Enable request for release by IRQ1-DS (P31)	Unused
			Enable request for release by IRQ2-DS (P32)	Unused
			Enable request for release by IRQ3-DS (P33)	Unused
			Enable request for release by IRQ4-DS (PB1)	Unused
			Enable request for release by IRQ5-DS (PA4)	Unused
			Enable request for release by IRQ6-DS (PA3)	Unused
			Enable request for release by IRQ7-DS (PE2)	Unused
			Enable request for release by IRQ8-DS (P40)	Unused
			Enable request for release by IRQ9-DS (P41)	Unused
			Enable request for release by IRQ10-DS (P42)	Unused
			Enable request for release by IRQ11-DS (P43)	Unused
			Enable request for release by IRQ12-DS (P44)	Unused
			Enable request for release by IRQ13-DS (P45)	Unused
			Enable request for release by IRQ14-DS (P46)	Unused

			Enable request for release by IRQ15-DS (P47)	Unused
			Enable request for release by NMI	Unused
			Enable request for release by SDA2-DS (P17)	Unused
			Enable request for release by SCL2-DS (P16)	Unused
			Enable request for release by LVD1	Unused
			Enable request for release by LVD2	Unused
			Enable request for release by CRX1-DS (P15)	Unused
			Enable request for release by the RTC periodic interrupt	Unused
			Enable request for release by the RTC alarm interrupt	Unused
			Enable request for release by USB suspension/resumption	Unused
			I/O port state retention	Simultaneous release from deep software standby mode and of retained I/O port state

[API setting example]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Enable IRQ5 interrupt */
    R_ICU_IRQ5_Start();

    /* Enable sleep mode */
    R_LPC_Sleep();

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_icu_user.c

```
/* Start user code for include. Do not edit comment generated here */
#include "r_cg_lpc.h"
/* End user code. Do not edit comment generated here */

static void r_icu_irq5_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Allow sleep mode return clock to be changed */
    R_LPC_ChangeSleepModeReturnClock(RETURN_MAIN_CLOCK);
    /* End user code. Do not edit comment generated here */
}
```

3.2.7 Buses

Below is a list of API functions output by the Code Generator for buses use.

Table 3.7 API Functions: [Buses]

API Function Name	Function
R_BSC_Create	Performs initialization necessary to control the buses.
R_BSC_Create_UserInit	Performs user-defined initialization relating to the buses.
r_bsc_buserr_interrupt	Performs processing in response to the bus error (illegal address access).
R_BSC_Error_Monitoring_Start	Allows the detection of bus errors (illegal address access).
R_BSC_Error_Monitoring_Stop	Prohibits the detection of bus errors (illegal address access).
R_BSC_InitializeSDRAM	Performs initialization of SDRAM controller.

R_BSC_Create

Performs initialization necessary to control the buses.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_BSC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_BSC_Create_UserInit

Performs user-defined initialization relating to the buses.

Remark This API function is called as the R_BSC_Create callback routine.

[Syntax]

```
void    R_BSC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_bsc_buserr_interrupt

Performs processing in response to the bus error (illegal address access).

Remark 1. This API function is called to run interrupt processing for a bus error (illegal address access), which is generated through access by the processing program to a location within an illegal address range.

Remark 2. The bus master that caused the bus error can be confirmed by reading the MST bit of bus error status register 1 (BERSR1) from within this API function.

Remark 3. The illegal address (high-order 13 bits) that caused the bus error can be confirmed by reading the ADDR bit of bus error status register 2 (BERSR2) from within this API function.

[Syntax]

```
static void     r_bsc_buserr_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_BSC_Error_Monitoring_Start

Allows the detection of bus errors (illegal address access).

[Syntax]

```
void    R_BSC_Error_Monitoring_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_BSC_Error_Monitoring_Stop

Prohibits the detection of bus errors (illegal address access).

[Syntax]

```
void    R_BSC_Error_Monitoring_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_BSC_InitializeSDRAM

Performs initialization of SDRAM controller.

[Syntax]

```
void    R_BSC_InitializeSDRAM ( void );
```

[Argument(s)]

None.

[Return value]

None.

Usage example

Get the address where bus error occurred.

[GUI setting example]

Buses				Used
	BSC0			Used
			Bus operation setting	Used
			On-Chip ROM setting	Enable
			ALE pin	Unused
			WAIT# pin	Unused
			BC1#/WR1# pin	Unused
			Memory bus 1 and 3 (RAM/Extended RAM) 0000 0000h to 0007 FFFFh 0080 0000h to 00FF FFFFh	The order of priority is fixed
			Memory bus 2 (ROM) 8000 0000h to FEFF FFFFh	The order of priority is fixed
			Internal peripheral bus 1 (Peripheral I/O registers) 0008 0000h to 0008 7FFFh	The order of priority is fixed
			Internal peripheral bus 2 and 3 (Peripheral I/O registers) 0008 8000h to 0009 FFFFh 000A 0000h to 000B FFFFh	The order of priority is fixed
			Internal peripheral bus 4 and 5 (Peripheral I/O registers) 000C 0000h to 000D FFFFh 000E 0000h to 000F FFFFh	The order of priority is fixed
			Internal peripheral bus 6 (E2 DataFlash memory and ROM) 0010 0000h to 00FF FFFFh	The order of priority is fixed
			External bus 0100 0000h to 0FFF FFFFh	The order of priority is fixed
			Separate bus CS recovery cycle insertion enable setting Read access after read access(Same area)	Unused
			Separate bus CS recovery cycle insertion enable setting Read access after read access(Different area)	Used
			Separate bus CS recovery cycle insertion enable setting Write access after read access(Same area)	Used
			Separate bus CS recovery cycle insertion enable setting Write access after read access(Different area)	Used
			Separate bus CS recovery cycle insertion enable setting Read access after write access(Same area)	Used

			Separate bus CS recovery cycle insertion enable setting Read access after write access(Different area)	Used
			Separate bus CS recovery cycle insertion enable setting Write access after write access(Same area)	Unused
			Separate bus CS recovery cycle insertion enable setting Write access after write access(Different area)	Unused
			Address/data multiplexed bus CS recovery cycle insertion enable setting Read access after read access(Same area)	Unused
			Address/data multiplexed bus CS recovery cycle insertion enable setting Read access after read access(Different area)	Used
			Address/data multiplexed bus CS recovery cycle insertion enable setting Write access after read access(Same area)	Used
			Address/data multiplexed bus CS recovery cycle insertion enable setting Write access after read access(Different area)	Used
			Address/data multiplexed bus CS recovery cycle insertion enable setting Read access after write access(Same area)	Used
			Address/data multiplexed bus CS recovery cycle insertion enable setting Read access after write access(Different area)	Used
			Address/data multiplexed bus CS recovery cycle insertion enable setting Write access after write access(Same area)	Unused
			Address/data multiplexed bus CS recovery cycle insertion enable setting Write access after write access(Different area)	Unused
			A7-A0, BC0#, DQM2, DQM3 (PA7-PA0)	Used
			A8	PB0
			A9	PB1
			A10	PB2
			A11	PB3
			A12	PB4
			A13	PB5
			A14	PB6
			A15	PB7

			A16	PC0
			A17	PC1
			A18	PC2
			A19	PC3
			A20	PC4
			A21	PC5
			A22	PC6
			A23	PC7
			Drive capacity setting	Unused
			Detect illegal address access	Used
			Detect timeout	Used
			Enable bus error interrupt (BUSERR)	Used
			Priority	Level 15 (highest)
		CS1		Used
			Use CS1 (0700 0000h to 07FF FFFFh)	Used
			CS1# output pin	P71
			Bus width	8 bits
			Endian	Same as operating mode
			Interface	Separate bus
			Write access mode	Byte strobe mode
			Page read access	Disable
			Page write access	Disable
			External wait	Disable
			Read recovery cycle	0 0 (ns)
			Write recovery cycle	0 0 (ns)
			Page read cycle wait	7 1166.666667 (ns)
			Page write cycle wait	7 1166.666667 (ns)
			Normal read cycle wait	7 1166.666667 (ns)
			Normal write cycle wait	7 1166.666667 (ns)
			Read-access CS extension cycle	7 1166.666667 (ns)
			Write-access CS extension cycle	0 0 (ns)
			Write data output extension cycle	0 0 (ns)
			Address cycle wait	0 0 (ns)
			RD assert wait	2 333.333333 (ns)
			WR assert wait	2 333.333333 (ns)
			Write data output wait	2 333.333333 (ns)
			CS assert wait	0 0 (ns)

[API setting example]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Enable BUSERR interrupt in ICU */
    R_BSC_Error_Monitoring_Start();

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_bsc_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint16_t g_bsc_buserr_addr;
/* End user code. Do not edit comment generated here */

static void r_bsc_buserr_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Restore an address that was accessed when a bus error occurred */
    if (1U == BSC.BERSR1.BIT.IA)
    {
        g_bsc_buserr_addr = ((uint16_t)(BSC.BERSR2.WORD)>>3U);
    }

    /* Clear the bus error status registers */
    BSC.BERCLR.BIT.STSCLR = 1U;
    /* End user code. Do not edit comment generated here */
}
```

3.2.8 DMA Controller (DMAC)

Below is a list of API functions output by the Code Generator for DMA controller use.

Table 3.8 API Functions: [DMA Controller]

API Function Name	Function
R_DMAC_Create	Performs initialization necessary to control the DMA controller.
R_DMACn_Start	Allows starting of the DMAC controller.
R_DMACn_Stop	Prohibits starting of the DMAC controller.
R_DMACn_Set_SoftwareTrigger	Sets the software request of DMA transfer by software.
R_DMACn_Clear_SoftwareTrigger	Clears the software request of DMA transfer by software.
r_dmac_dmacni_interrupt	Performs processing in response to the transfer end interrupt.
r_dmacn_callback_transfer_end	Performs processing in response to the transfer end interrupt.
r_dmacn_callback_transfer_escape_end	Performs processing in response to the escape transfer end interrupt.
R_DMAC_Create_UserInit	Performs user-defined initialization relating to the DMA controller.

R_DMAC_Create

Performs initialization necessary to control the DMA controller.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_DMAC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DMAC*n*_Start

Allows starting of the DMAC controller.

[Syntax]

```
void    R_DMACn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC*n*_Stop

Prohibits starting of the DMAC controller.

[Syntax]

```
void    R_DMACn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC*n*_Set_SoftwareTrigger

Sets the software request of DMA transfer by software.

[Syntax]

```
void    R_DMACn_Set_SoftwareTrigger ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC*n*_Clear_SoftwareTrigger

Clears the software request of DMA transfer by software.

[Syntax]

```
void    R_DMACn_Clear_SoftwareTrigger ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_dmac_dmac*n*i_interrupt

Performs processing in response to the transfer end interrupt.

Remark This API function is called as the interrupt process corresponding to the DMA transfer end interrupt for channel *n*

[Syntax]

```
static void    r_dmac_dmacni_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_dmac*n*_callback_transfer_end

Performs processing in response to the transfer end interrupt.

Remark This API function is called as thecall r_dmac_dmacni_interrupt back routine.

[Syntax]

```
static void    r_dmacn_callback_transfer_end ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_dmac*n*_callback_transfer_escape_end

Performs processing in response to the escape transfer end interrupt.

Remark This API function is called as the r_dmac_dmacni_interrupt callback routine, which is generated by escape transfer end.

[Syntax]

```
static void    r_dmacn_callback_transfer_escape_end ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC_Create_UserInit

Performs user-defined initialization relating to the DMA controller.

Remark This API function is called as the R_DMAC_Create callback routine.

[Syntax]

```
void    R_DMAC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

Usage example

The transfer starts by compare match interrupt, and flag is set at transfer completion.

[GUI setting example]

DMA Controller				Used
	Dmac			Used
		DmacChannel0		Used
			Activation source	CMT0 (CMI0 vect=28)
			Activation source flag control	Clear interrupt flag of the acitvation source
			Transfer mode	Normal mode
			Transfer data size	8 bits
			Transfer count	1
			Total transfer size	1byte(s)
			Source address	0x00000100(Fixed)
			Destination address	0x00000110(Fixed)
			Interrupt setting (DMAC0I)	Used
			Enable interrupt on transfer end	Used
			Priority	Level 15 (highest)

Compare Match Timer				Used
	CMT0			Used
			Compare match timer operation setting	Used
			Count clock setting	PCLK/32
			Interval value setting	100ms (Actual value：100)
			Enable compare match interrupt (CMI0)	Used
			Priority	Level 15 (highest)

[API setting example]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start CMT channel 0 counter */
    R_CMT0_Start();

    /* Enable the DMAC0 activation */
    R_DMAC0_Start();

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_dmac_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_dmac0_f;
/* End user code. Do not edit comment generated here */

void R_DMAC_Create_UserInit(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Clear the flag */
    g_dmac0_f = 0U;
    /* End user code. Do not edit comment generated here */
}

static void r_dmac0_callback_transfer_end(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Set the flag */
    g_dmac0_f = 1U;
    /* End user code. Do not edit comment generated here */
}
```

3.2.9 Data transfer controller (DTC)

Below is a list of API functions output by the Code Generator for data transfer controller use.

Table 3.9 API Functions: [Data Transfer Controller]

API Function Name	Function
R_DTC_Create	Performs initialization necessary to control the data transfer controller.
R_DTC_Create_UserInit	Performs user-defined initialization relating to the data transfer controller.
R_DTCm_Start	Allows starting of the data transfer controller.
R_DTCm_Stop	Prohibits starting of the data transfer controller.

R_DTC_Create

Performs initialization necessary to control the data transfer controller.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_DTC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DTC_Create_UserInit

Performs user-defined initialization relating to the data transfer controller.

Remark This API function is called as the R_DTC_Create callback routine.

[Syntax]

```
void    R_DTC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DTC*m*_Start

Allows starting the data transfer controller.

Remark In this API function, starting the data transfer controller is allowed by operating the DTCE bit of the DTC activation enable register *n* (DTCER*n*) supporting the transfer data number *m*.
m is the transfer data number, *n* is the interrupt vector number.

[Syntax]

```
void    R_DTCm_Start ( void );
```

Remark *m* is the transfer data number.

[Argument(s)]

None.

[Return value]

None.

R_DTC*m*_Stop

Prohibits starting of the data transfer controller.

Remark In this API function, starting the data transfer controller is prohibited by operating the DTCE bit of the DTC activation enable register *n* (DTCER*n*) supporting the transfer data number *m*.
m is the transfer data number, *n* is the interrupt vector number.

[Syntax]

```
void    R_DTCm_Stop ( void );
```

Remark *m* is the transfer data number.

[Argument(s)]

None.

[Return value]

None.

Usage example

The transfer starts by compare match interrupt.

[GUI setting example]

Data Transfer Controller				Used
	Dtc			Used
		BaseAddress		Used
			Transfer data read skip	Disable
			Address mode	Full-address mode (32 bits)
			DTC vector base address	0x0007FC00
		DtcChannel0		Used
			Transfer data 0	Used
			Chain transfer	Unused
			Activation source	CMT0 (CMI0 vect=28)
			Transfer mode setting	Normal mode
			Transfer data size setting	8 bits
			Interrupt setting	An interrupt request to the CPU is generated when specified data transfer is completed
			Source address	0x00000100(Address fixed)
			Destination address	0x00000110(Address fixed)
			Count	1
			Total transfer size	1 byte(s)

Compare Match Timer				Used
	CMT0			Used
			Compare match timer operation setting	Used
			Count clock setting	PCLK/32
			Interval value setting	100ms (Actual value : 100)
			Enable compare match interrupt (CMI0)	Used
			Priority	Level 15 (highest)

[API setting example]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start CMT channel 0 counter */
    R_CMT0_Start();

    /* Enable the DTC0 activation */
    R_DTC0_Start();

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

3.2.10 Event link controller (ELC)

Below is a list of API functions output by the Code Generator for event link controller use.

Table 3.10 API Functions: [Event Link Controller]

API Function Name	Function
R_ELC_Create	Performs initialization necessary to control the event link controller.
R_ELC_Create_UserInit	Performs user-defined initialization relating to the event link controller.
r_elc_elsrni_interrupt	Performs processing in response to the event link interrupt.
R_ELC_Start	Starts interlinked operation of peripheral functions.
R_ELC_Stop	Ends interlinked operation of peripheral functions.
R_ELC_GenerateSoftwareEvent	Generates the software event.
R_ELC_Set_PortBuffern	Sets the value of a port buffer.
R_ELC_Get_PortBuffern	Gets the value of a port buffer.

R_ELC_Create

Performs initialization necessary to control the event link controller.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_ELC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ELC_Create_UserInit

Performs user-defined initialization relating to the event link controller.

Remark This API function is called as the R_ELC_Create callback routine.

[Syntax]

```
void    R_ELC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_elc_elsr*n*i_interrupt

Performs processing in response to the event link interrupt.

Remark This API function is called to run interrupt processing for the event signal defined in event link setting register.
n is the source number.

[Syntax]

```
static void    r_elc_elsrni_interrupt ( void );
```

Remark *n* is the source number.

[Argument(s)]

None.

[Return value]

None.

R_ELC_Start

Starts interlinked operation of peripheral functions.

[Syntax]

```
void    R_ELC_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ELC_Stop

Ends interlinked operation of peripheral functions.

[Syntax]

```
void    R_ELC_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ELC_GenerateSoftwareEvent

Generates the software event.

[Syntax]

```
void    R_ELC_GenerateSoftwareEvent ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ELC_Set_PortBuffer*n*

Sets the value of a port buffer.

[Syntax]

```
void    R_ELC_Set_PortBuffer n ( uint8_t value );
```

Remark *n* is the source number.

[Argument(s)]

I/O	Argument	Description
I	uint8_t *value*;	The value set in the port buffer.

[Return value]

None.

R_ELC_Get_PortBuffer*n*

Gets the value of a port buffer.

[Syntax]

```
void    R_ELC_Get_PortBuffern ( uint8_t * const value );
```

Remark *n* is the source number.

[Argument(s)]

I/O	Argument	Description
O	uint8_t * const *value*;	Pointer to the location where the obtained value is to be stored.

[Return value]

None.

Usage example

Generate software events and generate linked events.
After linking one after another, when the ELC interruption comes, the event link is end.

[GUI setting example]

Event Link Controller				Used
	ELC			Used
		ELC_CMT1		Used
			CMT1	Used
			Event signal	Software event
			Operation on event	Counting is started
		ELC_Interrupt1		Used
			Interrupt 1	Used
			Event signal	Input edge detection of input port group 2
			Operation on event	Issues an event to the CPU
			ELSR18I Priority	Level 15 (highest)
		ELC_Interrupt2		Used
			Interrupt 2	Used
			Event signal	Input edge detection of input port group 2
			Operation on event	Issues an event to the CPU
			ELSR19I Priority	Level 15 (highest)
		PortGroup2		Used
			Port group 2 setting	Used
			PE0	Used
			PE1	Used
			PE2	Used
			PE3	Used
			PE4	Used
			PE5	Used
			PE6	Used
			PE7	Used
			Input port group 2 setting	Used
			Valid edge for event generation	Both
			Enable PDBF2 overwrite	Unused
			Input port group 2 setting Event signal	CMT1 compare match 1
			Input port group 2 setting Operation on event	Transfers the signal value of the external pin to the PDBFn register

Compare Match Timer				Used
	CMT1			Used
			Compare match timer operation setting	Used
			Count clock setting	PCLK/512
			Interval value setting	3000ms (Actual value：2999.978667)
			Enable compare match interrupt (CMI1)	Used
			Priority	Level 15 (highest)

[API setting example]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Enable all ELC event links */
    R_ELC_Start();

    /* Trigger a software event */
    R_ELC_GenerateSoftwareEvent();

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_elc_user.c

```
static void r_elc_elsr18i_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Disable all ELC event links */
    R_ELC_Stop();
    /* End user code. Do not edit comment generated here */
}
```

3.2.11 I/O ports

Below is a list of API functions output by the Code Generator for I/O ports use.

Table 3.11 API Functions: [I/O Ports]

API Function Name	Function
R_PORT_Create	Performs initialization necessary to control the I/O ports.
R_PORT_Create_UserInit	Performs user-defined initialization relating to the I/O ports.

R_PORT_Create

Performs initialization necessary to control the I/O ports.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_PORT_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PORT_Create_UserInit

Performs user-defined initialization relating to the I/O ports.

Remark This API function is called as the R_PORT_Create callback routine.

[Syntax]

```
void    R_PORT_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.12 Multi-function timer pulse unit 2 (MTU2)

Below is a list of API functions output by the Code Generator for multi-function timer pulse unit 2 use.

Table 3.12 API Functions: [Multi-Function Timer Pulse Unit 2]

API Function Name	Function
R_MTU2_Create	Performs initialization necessary to control the multi-function timer pulse unit 2.
R_MTU2_Create_UserInit	Performs user-defined initialization relating to the multi-function timer pulse unit 2.
r_mtu2_tgimn_interrupt	Performs processing in response to the input capture/compare match interrupt.
r_mtu2_cj_tgimn_interrupt	Performs processing in response to the input capture/compare match interrupt.
r_mtu2_tcivn_interrupt	Performs processing in response to the overflow interrupt.
r_mtu2_cj_tcivn_interrupt	Performs processing in response to the overflow interrupt.
r_mtu2_tciun_interrupt	Performs processing in response to the underflow interrupt.
R_MTU2_Cn_Start	Starts counting by a 16-bit timer.
R_MTU2_Cn_Stop	Ends counting by a 16-bit timer.

R_MTU2_Create

Performs initialization necessary to control the multi-function timer pulse unit 2.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_MTU2_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_MTU2_Create_UserInit

Performs user-defined initialization relating to the multi-function timer pulse unit 2.

Remark This API function is called as the R_MTU2_Create callback routine.

[Syntax]

```
void    R_MTU2_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_mtu2_tgi*mn*_interrupt

Performs processing in response to the input capture/compare match interrupt.

Remark This API function is called to run interrupt processing for the input capture interrupt generated because multi-function timer pulse unit 2 detected the effective edge of the input signal or for the compare match interrupt generated because the current counter value (value of the timer counter, TCNT) matched the defined counter value (value of the timer general register, TGR).

[Syntax]

```
static void    r_mtu2_tgimn_interrupt ( void );
```

Remark *m* is the timer general register number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_mtu2_c*j*_tgi*mn*_interrupt

Performs processing in response to the input capture/compare match interrupt.

Remark This API function is called to run interrupt processing for the input capture interrupt generated because multi-function timer pulse unit 2 detected the effective edge of the input signal or for the compare match interrupt generated because the current counter value (value of the timer counter, TCNT) matched the defined counter value (value of the timer general register, TGR).

[Syntax]

```
static void    r_mtu2_cj_tgimn_interrupt ( void );
```

Remark *j* is the relationship channel number, *m* is the timer general register number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_mtu2_tciv*n*_interrupt

Performs processing in response to the overflow interrupt.

Remark This API function is called to run interrupt processing for the overflow interrupt, which is generated in response to an overflow of the timer counter (TCNT).

[Syntax]

```
static void    r_mtu2_tciv n _interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_mtu2_c*j*_tciv*n*_interrupt

Performs processing in response to the overflow interrupt.

Remark This API function is called to run interrupt processing for the overflow interrupt, which is generated in response to an overflow of the timer counter (TCNT).

[Syntax]

```
static void    r_mtu2_cj_tcivn_interrupt ( void );
```

Remark *j* is the relationship channel number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_mtu2_tciu*n*_interrupt

Performs processing in response to the underflow interrupt.

Remark This API function is called to run interrupt processing for the underflow interrupt, which is generated in response to an underflow of the timer counter (TCNT).

[Syntax]

```
static void    r_mtu2_tciun_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_MTU2_C*n*_Start

Starts counting by the 16-bit timer.

[Syntax]

```
void    R_MTU2_Cn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_MTU2_C*n*_Stop

Ends counting by the 16-bit timer.

[Syntax]

```
void    R_MTU2_Cn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

Usage example

Use timer as One-shot timer.

[GUI setting example]

Multi-Function Timer Pulse Unit 2				Used
	MTU2_U0			Used
		MTU0		Used
			MTU0	Normal mode
			Include this channel in the synchronous operation	Unused
			Counter clock selection	PCLK
			Counter clear source	TGRA0 compare match/input capture (Use TGRA0 as a cycle register)
			TGRA0 (Output compare register)	50ms (Actual value：50)
			TGRB0 (Output compare register)	100μs (Actual value：100)
			TGRC0 (Output compare register)	100μs (Actual value：100)
			TGRD0 (Output compare register)	100μs (Actual value：100)
			TGRE0 (Output compare register)	100μs (Actual value：100)
			TGRF0 (Output compare register)	100μs (Actual value：100)
			MTIOC0A pin (PB3)	MTIOC0A pin output disabled
			MTIOC0B pin (P13)	MTIOC0B pin output disabled
			MTIOC0C pin (P32)	MTIOC0C pin output disabled
			MTIOC0D pin (P33)	MTIOC0D pin output disabled
			Enable A/D conversion start request on TGRA input capture/compare match (trigger signal of MTU0 TRGA0N)	Unused
			Enable TGRA0 input capture/compare match interrupt (TGIA0)	Used
			Enable TGRB0 input capture/compare match interrupt (TGIB0)	Unused
			Enable TGRC0 input capture/compare match interrupt (TGIC0)	Unused
			Enable TGRD0 input capture/compare match interrupt (TGID0)	Unused
			(TGIA/TGIB/TGIC/TGID) Priority	Level 15 (highest)
			Enable TGRE0 compare match interrupt (TGIE0)	Unused
			Enable TGRF0 compare match interrupt (TGIF0)	Unused
			Enable overflow interrupt (TCIV0)	Unused

[API setting example]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start MTU2 channel 0 counter */
    R_MTU2_C0_Start();

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_mtu2_user.c

```
static void r_mtu2_tgia0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop MTU2 channel 0 counter */
    R_MTU2_C0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

3.2.13 Multi-function timer pulse unit 3 (MTU3)

Below is a list of API functions output by the Code Generator for multi-function timer pulse unit 3 use.

Table 3.13 API Functions: [Multi-Function Timer Pulse Unit 3]

API Function Name	Function
R_MTU3_Create	Performs initialization necessary to control the multi-function timer pulse unit 3.
R_MTU3_Create_UserInit	Performs user-defined initialization relating to the multi-function timer pulse unit 3.
r_mtu3_tgimn_interrupt	Performs processing in response to the input capture/compare match interrupt.
r_mtu3_cj_tgimn_interrupt	Performs processing in response to the input capture/compare match interrupt.
r_mtu3_tcivn_interrupt	Performs processing in response to the overflow interrupt.
r_mtu3_cj_tcivn_interrupt	Performs processing in response to the overflow interrupt.
r_mtu3_tciun_interrupt	Performs processing in response to the underflow interrupt.
R_MTU3_Cn_Start	Starts counting by a 16-bit timer.
R_MTU3_Cn_Stop	Ends counting by a 16-bit timer.

R_MTU3_Create

Performs initialization necessary to control the multi-function timer pulse unit 3.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_MTU3_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_MTU3_Create_UserInit

Performs user-defined initialization relating to the multi-function timer pulse unit 3.

Remark This API function is called as the R_MTU3_Create callback routine.

[Syntax]

```
void    R_MTU3_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_mtu3_tgi*mn*_interrupt

Performs processing in response to the input capture/compare match interrupt.

Remark This API function is called to run interrupt processing for the input capture interrupt generated because multi-function timer pulse unit 3 detected the effective edge of the input signal or for the compare match interrupt generated because the current counter value (value of the timer counter, TCNT) matched the defined counter value (value of the timer general register, TGR).

[Syntax]

```
static void    r_mtu3_tgimn_interrupt ( void );
```

Remark *m* is the timer general register number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_mtu3_c*j*_tgi*mn*_interrupt

Performs processing in response to the input capture/compare match interrupt.

Remark This API function is called to run interrupt processing for the input capture interrupt generated because multi-function timer pulse unit 3 detected the effective edge of the input signal or for the compare match interrupt generated because the current counter value (value of the timer counter, TCNT) matched the defined counter value (value of the timer general register, TGR).

[Syntax]

```
static void     r_mtu3_cj_tgimn_interrupt ( void );
```

Remark *j* is the relationship channel number, *m* is the timer general register number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_mtu3_tciv*n*_interrupt

Performs processing in response to the overflow interrupt.

Remark This API function is called to run interrupt processing for the overflow interrupt, which is generated in response to an overflow of the timer counter (TCNT).

[Syntax]

```
static void    r_mtu3_tciv n _interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_mtu3_c*j*_tciv*n*_interrupt

Performs processing in response to the overflow interrupt.

Remark This API function is called to run interrupt processing for the overflow interrupt, which is generated in response to an overflow of the timer counter (TCNT).

[Syntax]

```
static void    r_mtu3_cj_tcivn_interrupt ( void );
```

Remark *j* is the relationship channel number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_mtu3_tciu*n*_interrupt

Performs processing in response to the underflow interrupt.

Remark This API function is called to run interrupt processing for the underflow interrupt, which is generated in response to an underflow of the timer counter (TCNT).

[Syntax]

```
static void    r_mtu3_tciun_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_MTU3_C*n*_Start

Starts counting by the 16-bit timer.

[Syntax]

```
void    R_MTU3_Cn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_MTU3_C*n*_Stop

Ends counting by the 16-bit timer.

[Syntax]

```
void    R_MTU3_Cn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

Usage example

Use timer as One-shot timer.

[GUI setting example]

Multi-Function Timer Pulse Unit 3				Used
	MTU3_U0			Used
			MTCLKA pin	Unused
			MTCLKB pin	Unused
			MTCLKC pin	Unused
			MTCLKD pin	Unused
		MTU0		Used
			MTU0	Normal mode
			Include this channel in the synchronous operation	Unused
			Counter clock selection	PCLK/64
			Clock edge setting	Rising edge
			Counter clear source	TGRA0 compare match/input capture (Use TGRA0 as a cycle register)
			TGRA0 (Output compare register)	50ms (Actual value：50.005333)
			TGRB0 (Output compare register)	100µs (Actual value：96)
			TGRC0 (Output compare register)	100µs (Actual value：96)
			TGRD0 (Output compare register)	100µs (Actual value：96)
			TGRE0 (Output compare register)	100µs (Actual value：96)
			TGRF0 (Output compare register)	100µs (Actual value：96)
			MTIOC0A pin (P34)	MTIOC0A pin output is disabled
			MTIOC0B pin (P13)	MTIOC0B pin output is disabled
			MTIOC0C pin (P32)	MTIOC0C pin output is disabled
			MTIOC0D pin (P33)	MTIOC0D pin output is disabled
			Enable A/D conversion start request on TGRA input capture/compare match (trigger signal of MTU0 TRGA0N)	Unused
			Enable A/D conversion start request on TGRE compare match (trigger signal of MTU0 TRG0N)	Unused
			Enable TGRA0 input capture/compare match interrupt (TGIA0)	Level 15 (highest)
			Enable TGRB0 input capture/compare match interrupt (TGIB0)	Unused
			Enable TGRC0 input capture/compare match interrupt (TGIC0)	Unused
			Enable TGRD0 input capture/compare match interrupt (TGID0)	Unused
			Enable TGRE0 compare match interrupt (TGIE0)	Unused
			Enable TGRF0 compare	Unused

			match interrupt (TGIF0)	
			Enable overflow interrupt (TCIV0)	Unused

[API setting example]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start MTU3 channel 0 counter */
    R_MTU3_C0_Start();

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_mtu3_user.c

```
static void r_mtu3_tgia0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop MTU3 channel 0 counter */
    R_MTU3_C0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

3.2.14 Port output enable 2 (POE2)

Below is a list of API functions output by the Code Generator for port output enable 2 use.

Table 3.14 API Functions: [Port Output Enable 2]

API Function Name	Function
R_POE2_Create	Performs initialization necessary to control the port output enable 2.
R_POE2_Create_UserInit	Performs user-defined initialization relating to the port output enable 2.
r_poe2_oein_interrupt	Performs processing in response to the output enable interrupt n (OEI*n*).
R_POE2_Start	Places the MTU's complementary PWM output pins in the high-impedance state.
R_POE2_Stop	Releases the R_POE2_Stop MTU's complementary PWM output pins from the high-impedance state.
R_POE2_Set_HiZ_MTUn	Sets the high-impedance state for the MTU*n* pins.
R_POE2_Clear_HiZ_MTUn	Clear the high-impedance state for the MTU*n* pins

R_POE2_Create

Performs initialization necessary to control the port output enable 2.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_POE2_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_POE2_Create_UserInit

Performs user-defined initialization relating to the port output enable 2.

Remark This API function is called as the R_POE2_Create callback routine.

[Syntax]

```
void    R_POE2_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_poe2_oei*n*_interrupt

Performs processing in response to the output enable interrupt n (OEI*n*).

Remark This API function is called to run interrupt processing for the output enable interrupt n (OEI*n*), which is generated when a pin (any of POE0#, POE1#, POE2#, POE3#, and POE8#) becomes high-impedance or the output short flag 1 is set.

[Syntax]

```
static void    r_poe2_oein_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_POE2_Start

Places the MTU's complementary PWM output pins in the high-impedance state.

[Syntax]

```
void    R_POE2_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_POE2_Stop

Releases the MTU's complementary PWM output pins from the high-impedance state.

[Syntax]

```
void    R_POE2_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_POE2_Set_HiZ_MTU*n*

Sets the high-impedance state for the MTU*n* pins.

[Syntax]

```
void    R_POE2_Set_HiZ_MTUn ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_POE2_Clear_HiZ_MTU*n*

Clear the high-impedance state for the MTU*n* pins.

[Syntax]

```
void    R_POE2_Clear_HiZ_MTUn ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

Usage example

Sets the high-impedance state for the MTU0 pins by output enable interrupt.

[GUI setting example]

Port Output Enable 2				Used
	POE2			Used
			Enable MTIOC0A	Used
			Enable MTIOC0B	Used
			Enable MTIOC0C	Used
			Enable MTIOC0D	Used
			POE8# pin	P17
			POE8 mode select accepts a request	On the falling edge of POE8# input
			Output enable interrupt 2 (OEI2)	Used
			OEI2 Priority	Level 15 (highest)
			Enable MTIOC4A and MTIOC4C	MTIOC4C
			Enable MTIOC4B and MTIOC4D	MTIOC4D
			Output enable interrupt 1 (OEI1)	Unused
			Enable request to place pins in the high-impedance on detection of stopped oscillation	Unused

Multi-Function Timer Pulse Unit 2				Used
	MTU2_U0			Used
		MTU0		Used
			MTU0	PWM mode 1
			Include this channel in the synchronous operation	Unused
			Counter clock selection	PCLK
			Counter clear source	Disabled counter clear
			TGRC0	Output compare register
			TGRD0	Output compare register
			TGRE0	Output compare register
			TGRF0	Output compare register
			MTIOC0A pin (Initial output of MTIOC0A pin is 0. 0 output at compare match.)	P34
			When TGRB compare match	0 output from MTIOC0A pin
			MTIOC0C pin (Initial output of MTIOC0C pin is 0. 0 output at compare match.)	P32
			When TGRD compare match	0 output from MTIOC0C pin
			Initial of compare match A value (TGRA)	100
			Initial of compare match B value (TGRB)	100
			Initial of compare match C	100

			value (TGRC)	
			Initial of compare match D value (TGRD)	100
			Initial of compare match E value (TGRE)	100
			Initial of compare match F value (TGRF)	100
			Enable A/D conversion start request on TGRA input capture/compare match (trigger signal of MTU0 TRGA0N)	Unused
			Enable TGRA0 input capture/compare match interrupt (TGIA0)	Unused
			Enable TGRB0 input capture/compare match interrupt (TGIB0)	Unused
			Enable TGRC0 input capture/compare match interrupt (TGIC0)	Unused
			Enable TGRD0 input capture/compare match interrupt (TGID0)	Unused
			Enable TGRE0 compare match interrupt (TGIE0)	Unused
			Enable TGRF0 compare match interrupt (TGIF0)	Unused
			Enable overflow interrupt (TCIV0)	Unused

[API setting example]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start the POE2 module */
    R_POE2_Start();

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_poe2_user.c

```
static void r_poe2_oei2_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop the POE2 module */
    R_POE2_Stop();
    /* End user code. Do not edit comment generated here */
}
```

3.2.15 Port output enable 3 (POE3)

Below is a list of API functions output by the Code Generator for port output enable 3 use.

Table 3.15 API Functions: [Port Output Enable 3]

API Function Name	Function
R_POE3_Create	Performs initialization necessary to control the port output enable 3.
R_POE3_Create_UserInit	Performs user-defined initialization relating to the port output enable 3.
r_poe3_oein_interrupt	Performs processing in response to the output enable interrupt n (OEI*n*).
R_POE3_Start	Places the MTU's complementary PWM output pins in the high-impedance state.
R_POE3_Stop	Releases the R_POE3_Stop MTU's complementary PWM output pins from the high-impedance state.
R_POE3_Set_HiZ_MTUn	Sets the high-impedance state for the MTU*n* pins.
R_POE3_Clear_HiZ_MTUn	Clear the high-impedance state for the MTU*n* pins.
R_POE3_Set_HiZ_GPTn	Sets the high-impedance state for the GPT*n* pins.
R_POE3_Clear_HiZ_GPTn	Clear the high-impedance state for the GPT*n* pins.

R_POE3_Create

Performs initialization necessary to control the port output enable 3.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_POE3_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_POE3_Create_UserInit

Performs user-defined initialization relating to the port output enable 3.

Remark This API function is called as the R_POE3_Create callback routine.

[Syntax]

```
void    R_POE3_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_poe3_oei*n*_interrupt

Performs processing in response to the output enable interrupt.

Remark This API function is called to run interrupt processing for the output enable interrupt, which is generated when a related pin becomes high-impedance or the output short flag 1 is set.

[Syntax]

```
static void    r_poe3_oein_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_POE3_Start

Places the related pins in the high-impedance state.

[Syntax]

```
void    R_POE3_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_POE3_Stop

Replaces the related output pins from the high-impedance state.

[Syntax]

```
void    R_POE3_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_POE3_Set_HiZ_MTU*n*

Sets the high-impedance state for the MTU*n* pins.

[Syntax]

```
void    R_POE3_Set_HiZ_MTUn ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_POE3_Clear_HiZ_MTU*n*

Clear the high-impedance state for the MTU*n* pins.

[Syntax]

```
void    R_POE3_Clear_HiZ_MTUn ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_POE3_Set_HiZ_GPT*n*

Sets the high-impedance state for the GPT*n* pins.

[Syntax]

```
void    R_POE3_Set_HiZ_GPTn ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_POE3_Clear_HiZ_GPT*n*

Clear the high-impedance state for the GPT*n* pins.

[Syntax]

```
void    R_POE3_Clear_HiZ_GPTn ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

Usage example

Sets the high-impedance state for the MTU0 pins by output enable interrupt.

[GUI setting example]

Port Output Enable 3				Used
	POE3			Used
			MTU0 output pin control setting	
			Enable MTIOC0A	Used
			MTIOC0A pin	Controls high-impedance state of P34
			Enable MTIOC0B	Unused
			Enable MTIOC0C	Used
			MTIOC0C pin	Controls high-impedance state of P32
			Enable MTIOC0D	Unused
			POE0# input level detection (MTU0)	Unused
			POE4# input level detection (MTU0)	Unused
			POE10# input level detection (MTU0)	Unused
			POE11# input level detection (MTU0)	Unused
			Enable POE8 input	Used
			POE8# pin	P17
			POE8 mode select accepts a request	On the falling edge of POE8# input
			Output enable interrupt 3 (OEI3)	Used
			OEI3 Priority (Group BL1)	Level 15 (highest)
			MTU3 and MTU4/GPT0 GPT1 and GPT2 output pin control setting	
			MTU3/GPT0 select	Unused
			MTU4/GPT1 select	Unused
			MTU4/GPT2 select	Unused
			Enable POE0 input	Unused
			Output enable interrupt 1 (OEI1)	Unused
			MTU6 and MTU7 output pin control setting	
			Enable MTIOC6B and MTIOC6D	Unused
			Enable MTIOC7A and MTIOC7C	Unused
			Enable MTIOC7B and MTIOC7D	Unused
			Enable POE4 input	Unused
			Output enable interrupt 2 (OEI2)	Unused
			GPT0 GPT1 GPT2 and GPT3 output pin control setting	
			Enable GTIOC0A and GTIOC0B	Unused
			Enable GTIOC1A and GTIOC1B	Unused
			Enable GTIOC2A and GTIOC2B	Unused

			Enable GTIOC3A and GTIOC3B	Unused
			Enable POE10 input	Unused
			Enable POE11 input	Unused
			Output enable interrupt 4 (OEI4)	Unused
			Detection of stopped oscillation setting	
			Enable request to place pins in the high-impedance on detection of stopped oscillation	Unused

Interrupt Controller Unit				Used
	ICU			Used
		Group		Used
			Group BL1	Used
			Group BL1 Priority	Level 15 (highest)

Multi-Function Timer Pulse Unit 3				Used
	MTU3_U0			Used
			MTCLKA pin	Unused
			MTCLKB pin	Unused
			MTCLKC pin	Unused
			MTCLKD pin	Unused
		MTU0		Used
			MTU0	Normal mode
			Include this channel in the synchronous operation	Unused
			Counter clock selection	PCLK
			Counter clear source	Disabled counter clear
			TGRA0 (Output compare register)	100μs (Actual value：100)
			TGRB0 (Output compare register)	100μs (Actual value：100)
			TGRC0 (Output compare register)	100μs (Actual value：100)
			TGRD0 (Output compare register)	100μs (Actual value：100)
			TGRE0 (Output compare register)	100μs (Actual value：100)
			TGRF0 (Output compare register)	100μs Actual value：100)
			MTIOC0A pin (P34)	Initial output of MTIOC0A pin is 0. 0 output at compare match.
			MTIOC0B pin (P13)	MTIOC0B pin output is disabled
			MTIOC0C pin (P32)	Initial output of MTIOC0C pin is 0. 0 output at compare match.
			MTIOC0D pin (P33)	MTIOC0D pin output is disabled
			Enable A/D conversion start request on TGRA input capture/compare match (trigger signal of MTU0 TRGA0N)	Unused
			Enable A/D conversion start request on TGRE compare match (trigger signal of MTU0 TRG0N)	Unused

			Enable TGRA0 input capture/compare match interrupt (TGIA0)	Unused
			Enable TGRB0 input capture/compare match interrupt (TGIB0)	Unused
			Enable TGRC0 input capture/compare match interrupt (TGIC0)	Unused
			Enable TGRD0 input capture/compare match interrupt (TGID0)	Unused
			Enable TGRE0 compare match interrupt (TGIE0)	Unused
			Enable TGRF0 compare match interrupt (TGIF0)	Unused
			Enable overflow interrupt (TCIV0)	Unused

[API setting example]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start the POE3 module */
    R_POE3_Start();

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_poe3_user.c

```
void r_poe3_oei3_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop the POE3 module */
    R_POE3_Stop();
    /* End user code. Do not edit comment generated here */
}
```

3.2.16 General PWM timer (GPT)

Below is a list of API functions output by the Code Generator for general PWM timer use.

Table 3.16 API Functions: [General PWM timer]

API Function Name	Function
R_GPT_Create	Performs initialization necessary to control the general PWM timer.
R_GPTn_Start	Starts counting by a 16-bit timer.
R_GPTn_Stop	Ends counting by a 16-bit timer.
R_GPTn_HardwareStart	Allows GPT interrupts.
R_GPTn_HardwareStop	Prohibits GPT interrupts.
R_GPT_Create_UserInit	Performs user-defined initialization relating to the general PWM timer.
r_gpt_gtcimn_interrupt	Performs processing in response to the input capture/compare match interrupt.
r_gpt_gtcivn_interrupt	Performs processing in response to the overflow interrupt.
r_gpt_gtciun_interrupt	Performs processing in response to the underflow interrupt.
r_gpt_gdten_interrupt	Performs processing in response to the dead time error interrupt.
r_gpt_etgip_interrupt	Performs processing in response to the external trigger rising interrupt.
r_gpt_etgin_interrupt	Performs processing in response to the external trigger falling interrupt.

R_GPT_Create

Performs initialization necessary to control the general PWM timer.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_GPT_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_GPT*n*_Start

Starts counting by a 16-bit timer.

[Syntax]

```
void    R_GPTn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_GPT*n*_Stop

Ends counting by a 16-bit timer.

[Syntax]

```
void    R_GPTn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_GPT*n*_HardwareStart

Allows detection of GPT interrupts.

Remark This API function enables GPT interrupts when starting timer count by the hardware trigger.

[Syntax]

```
void R_GPTn_HardwareStart ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_GPT*n*_HardwareStop

Prohibits detection of GPT interrupts.

Remark This API function disables GPT interrupts when starting timer count by the hardware trigger.

[Syntax]

```
void R_GPTn_HardwareStop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_GPT_Create_UserInit

Performs user-defined initialization relating to the general PWM timer.

Remark This API function is called as the R_GPT_Create callback routine.

[Syntax]

```
void    R_GPT_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_gpt_gtci*mn*_interrupt

Performs processing in response to the input capture/compare match interrupt.

[Syntax]

```
static void    r_gpt_gtcimn_interrupt ( void );
```

Remark *m* is the timer general register number, *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_gpt_gtciv*n*_interrupt

Performs processing in response to the overflow interrupt.

[Syntax]

```
static void     r_gpt_gtciv n _interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_gpt_gtciu*n*_interrupt

Performs processing in response to the underflow interrupt.

[Syntax]

```
static void     r_gpt_gtciun_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_gpt_gdte*n*_interrupt

Performs processing in response to the dead time error interrupt.

[Syntax]

```
static void    r_gpt_gdten_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_gpt_etgip_interrupt

Performs processing in response to the external trigger rising interrupt.

[Syntax]

```
static void     r_gpt_etgip_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_gpt_etgin_interrupt

Performs processing in response to the external trigger falling interrupt.

[Syntax]

```
static void     r_gpt_etgin_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

Usage example

Use timer as One-shot timer.

[GUI setting example]

General PWM Timer				Used
	Gpt0			Used
			GTETRG pin	Unused
		GptChannel0		Used
			GPT0	Saw-wave one-shot pulse mode
			Clock source	PCLKA/8 0.75 (MHz)
			Timer operation period	50 ms (Actual value: 50)
			Period register value (GTPR0)	37499
			Buffer operation(GTPR)	Buffer operation is not performed
			Count direction	Up-counting
			Counter initial value	0
			Counter clear	Disable
			Hardware source for counter start	Disable
			Hardware source for counter stop/clear	Disable
			GTCCRA operation	Compare match
			Compare match value (GTCCRA)	10
			Buffer operation(GTCCRA)	Double buffer operation
			GTIOC0A pin function	Disable
			GTCCRB operation	Compare match
			Compare match value (GTCCRB)	20
			Buffer operation(GTCCRB)	Double buffer operation
			GTIOC0B pin function	Disable
			Automatically set GTCCRB0 using GTCCRA0 value and dead time	Unused
			GTCCRC operation	Buffer register for GTCCRA
			GTCCRD operation	Double buffer register for GTCCRA
			GTCCRE operation	Buffer register for GTCCRB
			GTCCRF operation	Double buffer register for GTCCRB
			Enable compare match (up-counting) A/D conversion start request(GTADTRA)	Unused
			Enable compare match (down-counting) A/D conversion start request(GTADTRA)	Unused
			Enable compare match (up-counting) A/D conversion start request(GTADTRB)	Unused
			Enable compare match (down-counting) A/D conversion start request(GTADTRB)	Unused

			Enable GTCCRA input capture/compare match interrupt (GTCIA0)	Used Level 15 (highest)
			Enable GTCCRB input capture/compare match interrupt (GTCIB0)	Used Level 15 (highest)
			Enable GTCNT overflow (GTPR compare match) interrupt (GTCIV0)	Used Level 15 (highest)

[API setting example]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start GPT channel 0 counter */
    R_GPT0_Start();

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_gpt_user.c

```
static void r_gpt_gtciv0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop GPT channel 0 counter */
    R_GPT0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

3.2.17 16-bit timer pulse unit (TPU)

Below is a list of API functions output by the Code Generator for 16-bit timer pulse unit use.

Table 3.17 API Functions: [16-bit timer pulse unit]

API Function Name	Function
R_TPU_Create	Performs initialization necessary to control the 16-bit timer pulse unit.
R_TPUn_Start	Starts counting by a 16-bit timer.
R_TPUn_Stop	Ends counting by a 16-bit timer.
R_TPU_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer pulse unit.
r_tpu_tginm_interrupt	Performs processing in response to the input capture/compare match interrupt.
r_tpu_tcinv_interrupt	Performs processing in response to the overflow interrupt.
r_tpu_tcinu_interrupt	Performs processing in response to the underflow interrupt.

R_TPU_Create

Performs initialization necessary to control the 16-bit timer pulse unit.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_TPU_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TPU*n*_Start

Starts counting by a 16-bit timer.

[Syntax]

```
void    R_TPUn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TPU*n*_Stop

Ends counting by a 16-bit timer.

[Syntax]

```
void    R_TPUn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TPU_Create_UserInit

Performs user-defined initialization relating to the 16-bit timer pulse unit.

Remark This API function is called as the R_TPU_Create callback routine.

[Syntax]

```
void    R_TPU_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_tpu_tgi*nm*_interrupt

Performs processing in response to the input capture/compare match interrupt.

[Syntax]

```
static void    r_tpu_tgi nm _interrupt ( void );
```

Remark *m* is the timer general register number, *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_tpu_tci*n*v_interrupt

Performs processing in response to the overflow interrupt.

[Syntax]

```
static void    r_tpu_tci n v_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_tpu_tci*n*u_interrupt

Performs processing in response to the underflow interrupt.

[Syntax]

```
static void    r_tpu_tci n u_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

Usage example

Use timer as One-shot timer.

[GUI setting example]

16-Bit Timer Pulse Unit				Used
	TPU_U0			Used
			TCLKA pin	Unused
			TCLKB pin	Unused
			TCLKC pin	Unused
			TCLKD pin	Unused
		TPU0		Used
			Include this channel in the synchronous operation	Unused
			Counter clock selection	PCLK/64
			Clock edge setting	Rising edge
			Counter clear source	TGRA0 compare match/input capture (Use TGRA0 as a cycle register)
			TGRA0 (Output compare register)	100ms (Actual value：100)
			TGRB0 (Output compare register)	100μs (Actual value：96)
			TGRC0 (Output compare register)	100μs (Actual value：96)
			TGRD0 (Output compare register)	100μs (Actual value：96)
			TIOCA0 pin (PA0)	TIOCA0 pin output is disabled
			TIOCB0 pin (PA1)	TIOCB0 pin output is disabled
			TIOCC0 pin (P32)	TIOCC0 pin output is disabled
			TIOCD0 pin (PA3)	TIOCD0 pin output is disabled
			Enable A/D conversion start request on TGRA input capture/compare match (trigger signal of TPU0 TRGA0N)	Unused
			Enable TGRA0 input capture/compare match interrupt (TGI0A)	Level 15 (highest)
			Enable TGRB0 input capture/compare match interrupt (TGI0B)	Unused
			Enable TGRC0 input capture/compare match interrupt (TGI0C)	Unused
			Enable TGRD0 input capture/compare match interrupt (TGI0D)	Unused
			Enable overflow interrupt (TCI0V)	Unused

[API setting example]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start TPU channel 0 counter */
    R_TPU0_Start();

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_tpu_user.c

```
static void r_tpu_tgi0a_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop TPU channel 0 counter */
    R_TPU0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

3.2.18 8-bit timer (TMR)

Below is a list of API functions output by the Code Generator for 8-bit timer use.

Table 3.18 8 API Functions: [8-bit timer]

API Function Name	Function
R_TMR_Create	Performs initialization necessary to control the 8-bit timer pulse unit.
R_TMRn_Start	Starts counting by an 8-bit timer.
R_TMRn_Stop	Ends counting by an 8-bit timer.
R_TMR_Create_UserInit	Performs user-defined initialization relating to the 8-bit timer pulse unit.
r_tmr_cmimn_interrupt	Performs processing in response to the compare match interrupt.
r_tmr_ovin_interrupt	Performs processing in response to the overflow interrupt.

R_TMR_Create

Performs initialization necessary to control the 8-bit timer.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_TMR_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR*n*_Start

Starts counting by an 8-bit timer.

[Syntax]

```
void    R_TMRn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR*n*_Stop

Ends counting by an 8-bit timer.

[Syntax]

```
void    R_TMRn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_Create_UserInit

Performs user-defined initialization relating to the 8-bit timer.

Remark This API function is called as the R_TMR_Create callback routine.

[Syntax]

```
void    R_TMR_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_tmr_cmi*mn*_interrupt

Performs processing in response to the compare match interrupt.

[Syntax]

```
static void     r_tmr_cmimn_interrupt ( void );
```

Remark *m* is the timer general register number, *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_tmr_ovi*n*_interrupt

Performs processing in response to the overflow interrupt.

[Syntax]

```
static void     r_tmr_ovin_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

Usage example

Use timer as One-shot timer.

[GUI setting example]

8-Bit Timer				Used
	Tmr0			Used
		TmrChannel0		Used
			TMR0	8-bit count mode
			Clock source	PCLK/8192 0.732 (kHz)
			Counter clear	Cleared by compare match A
			Compare match A value (TCORA)	20 ms (Actual value: 20.48)
			S12AD A/D conversion start request	Unused
			Compare match B value (TCORB)	20 ms (Actual value: 20.48)
			Enable TMO0 output	Unused
			Enable TCORA compare match interrupt (CMIA0)	Used Level 15 (highest)
			Enable TCORB compare match interrupt (CMIB0)	Unused
			Enable TCNT overflow interrupt (OVI0)	Unused

[API setting example]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start TMR channel 0 counter */
    R_TMR0_Start();

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_tmr_user.c

```
static void r_tmr_cmia0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop TMR channel 0 counter */
    R_TMR0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

3.2.19 Programmable pulse generator (PPG)

Below is a list of API functions output by the Code Generator for programmable pulse generator use.

Table 3.19 API Functions: [Programmable Pulse Generator]

API Function Name	Function
R_PPG_Create	Performs initialization necessary to control the programmable pulse generator.
R_PPG_Create_UserInit	Performs user-defined initialization relating to the programmable pulse generator.

R_PPG_Create

Performs initialization necessary to control the programmable pulse generator.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_PPG_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PPG_Create_UserInit

Performs user-defined initialization relating to the programmable pulse generator.

Remark This API function is called as the R_PPG_Create callback routine.

[Syntax]

```
void    R_PPG_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.2.20 Compare match timer (CMT)

Below is a list of API functions output by the Code Generator for compare match timer use.

Table 3.20 API Functions: [Compare Match Timer]

API Function Name	Function
R_CMTn_Create	Performs initialization necessary to control the compare match timer.
R_CMTn_Create_UserInit	Performs user-defined initialization relating to the compare match timer.
r_cmt_cmin_interrupt	Performs processing in response to the compare match interrupt (CMI*n*).
R_CMTn_Start	Starts counting by a 16-bit timer.
R_CMTn_Stop	Ends counting by a 16-bit timer.

R_CMT*n*_Create

Performs initialization necessary to control the compare match timer.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_CMTn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CMT*n*_Create_UserInit

Performs user-defined initialization relating to the compare match timer.

Remark This API function is called as the R_CMTn_Create callback routine.

[Syntax]

```
void    R_CMTn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_cmt_cmi*n*_interrupt

Performs processing in response to the compare match interrupt (CMI*n*).

Remark This API function is called to run interrupt processing for the compare match interrupt (CMI*n*), which is generated because the current counter value (value of the compare match timer counter, CMCR) matched the defined counter value (value of the compare match timer constant register, CMCOR).

[Syntax]

```
static void    r_cmt_cmin_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CMT*n*_Start

Starts counting by a 16-bit timer.

[Syntax]

```
void    R_CMTn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CMT*n*_Stop

Ends counting by a 16-bit timer.

[Syntax]

```
void    R_CMTn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

Usage example

Use timer as One-shot timer.

[GUI setting example]

Compare Match Timer				Used
	CMT0			Used
			Compare match timer operation setting	Used
			Count clock setting	PCLK/512
			Interval value setting	100ms (Actual value：100.010667)
			Enable compare match interrupt (CMI0)	Used
			Priority	Level 15 (highest)

[API setting example]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start CMT channel 0 counter */
    R_CMT0_Start();

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_cmt_user.c

```
static void r_cmt_cmi0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop CMT channel 0 counter */
    R_CMT0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

3.2.21 Compare match timer W (CMTW)

Below is a list of API functions output by the Code Generator for compare match timer W use.

Table 3.21 API Functions: [Compare Match Timer W]

API Function Name	Function
R_CMTWn_Create	Performs initialization necessary to control the compare match timer W.
R_CMTWn_Start	Starts counting by a 16-bit timer.
R_CMTWn_Stop	Ends counting by a 16-bit timer.
R_CMTWn_Create_UserInit	Performs user-defined initialization relating to the compare match timer W.
r_cmtw_cmwin_interrupt	Performs processing in response to the compare match interrupt.
r_cmtw_icmin_interrupt	Performs processing in response to the input capture interrupt.
r_cmtw_ocmin_interrupt	Performs processing in response to the output compare interrupt.

R_CMTW*n*_Create

Performs initialization necessary to control the compare match timer W.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_CMTWn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CMTW*n*_Start

Starts counting by a 16-bit timer.

[Syntax]

```
void    R_CMTWn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CMTW*n*_Stop

Ends counting by a 16-bit timer.

[Syntax]

```
void    R_CMTWn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CMTW*n*_Create_UserInit

Performs user-defined initialization relating to the compare match timer W.

Remark This API function is called as the R_CMTWn_Create callback routine.

[Syntax]

```
void R_CMTWn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_cmtw_cmwi*n*_interrupt

Performs processing in response to the compare match interrupt.

[Syntax]

```
static void     r_cmtw_cmwin_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_cmtw_ic*min*_interrupt

Performs processing in response to the input capture interrupt.

[Syntax]

```
static void     r_cmtw_icmin_interrupt ( void );
```

Remark *m* is the timer general register number, *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_cmtw_oc*m*i*n*_interrupt

Performs processing in response to the output compare interrupt.

[Syntax]

```
static void     r_cmtw_oc*m*i*n*_interrupt ( void );
```

Remark *m* is the timer general register number, *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

Usage example

Use timer as One-shot timer.

[GUI setting example]

Compare Match Timer W				Used
	CMTW0			Used
			Compare match timer W operation setting	Used
			Count Clock setting	PCLK/8
			Timer Counter size	32 bits
			Compare match setting Interval value	100ms (Actual value：100)
			Register value (CMWCOR)	74999
			Compare match interrupt (CMWI0)	Used
			Priority (CMWI0)	Level 15 (highest)
			Counter clear	CMWCNT is cleared by CMWCOR
			TIC0 pin	Used
			Input capture control 0	Rising edge
			Enable input capture interrupt (IC0I0)	Used
			Priority (IC0I0)	Level 15 (highest)
			TIC1 pin	Used
			Input capture control 1	Rising edge
			Enable input capture interrupt (IC1I0)	Used
			Priority (IC1I0)	Level 15 (highest)
			TOC0 pin	Used
			Output compare control 0	Retains the output value
			Compare value 0	10
			Enalbe output compare interrupt (OC0I0)	Used
			Priority (OC0I0)	Level 15 (highest)
			TOC1 pin	Used
			Output compare control 1	Retains the output value
			Compare value 1	20
			Enable output compare interrupt (OC1I0)	Used
			Priority (OC1I0)	Level 15 (highest)

[API setting example]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start CMTW channel 0 counter */
    R_CMTW0_Start();

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_cmtw_user.c

```
static void r_cmtw_cmwi0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop CMTW channel 0 counter */
    R_CMTW0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

3.2.22 Realtime clock (RTC)

Below is a list of API functions output by the Code Generator for realtime clock use.

Table 3.22 API Functions: [Realtime Clock]

API Function Name	Function
R_RTC_Create	Performs initialization necessary to control the realtime clock.
R_RTC_Create_UserInit	Performs user-defined initialization relating to the realtime clock.
r_rtc_alm_interrupt	Performs processing in response to the alarm interrupt (ALM).
r_rtc_prd_interrupt	Performs processing in response to the periodic interrupt (PRD).
r_rtc_cup_interrupt	Performs processing in response to the carry interrupt (CUP).
R_RTC_Set_CalendarAlarm	Sets the condition for the alarm interrupt (ALM) and allows detection of ALM (calendar count mode).
R_RTC_Set_BinaryAlarm	Sets the condition for the alarm interrupt (ALM) and allows detection of ALM (binary count mode).
R_RTC_Set_ConstPeriodInterruptOn	Sets the period of the periodic interrupt (PRD) and allows detection of PRD.
R_RTC_Set_ConstPeriodInterruptOff	Prohibits detection of the periodic interrupt (PRD).
R_RTC_Set_CarryInterruptOn	Allows detection of the carry interrupt (CUP).
R_RTC_Set_CarryInterruptOff	Prohibits detection of the carry interrupt (CUP).
R_RTC_Set_RTCOUTOn	Set the RTCOUT output period and starts RTCOUT output.
R_RTC_Set_RTCOUTOff	Ends the RTCOUT output.
R_RTC_Start	Starts counting.
R_RTC_Stop	Ends counting.
R_RTC_Restart	Initializes the counter then starts counting.
R_RTC_Set_CalendarCounterValue	Sets the values of the calendar and time counters.
R_RTC_Get_CalendarCounterValue	Gets the values of the calendar and time counters.
R_RTC_Set_BinaryCounterValue	Sets the value of the binary counter.
R_RTC_Get_BinaryCounterValue	Gets the value of the binary counter.
R_RTC_Get_CalendarTimeCaptureValuen	Gets the captured calendar time value.
R_RTC_Get_BinaryTimeCaptureValuen	Gets the captured binary time value.

R_RTC_Create

Performs initialization necessary to control the realtime clock.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_RTC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Create_UserInit

Performs user-defined initialization relating to the realtime clock.

Remark This API function is called as the R_RTC_Create callback routine.

[Syntax]

```
void    R_RTC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_rtc_alm_interrupt

Performs processing in response to the alarm interrupt (ALM).

Remark This API function is called to run interrupt processing for the alarm interrupt (ALM), which is generated when the condition specified by R_RTC_Set_CalendarAlarm is satisfied.

[Syntax]

```
static void     r_rtc_alm_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_rtc_prd_interrupt

Performs processing in response to the periodic interrupt (PRD).

Remark This API function is called to run interrupt processing for the periodic interrupt (PRD), which is generated when the period specified by R_RTC_Set_ConstPeriodInterruptOn elapses.

[Syntax]

```
static void     r_rtc_prd_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_rtc_cup_interrupt

Performs processing in response to the carry interrupt (CUP).

Remark This API function is called to run interrupt processing for the carry interrupt (CUP), which is generated on carries from the seconds counter (RSECCNT) or binary counter 0 (BCNT0) or when the 64-Hz counter (R64CNT) is read at the same time as a carry from the 64-Hz counter (R64CNT).

[Syntax]

```
static void     r_rtc_cup_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_CalendarAlarm

Sets the condition for the alarm interrupt (ALM) and allows detection of ALM (calendar count mode).

[Syntax]

```
#include          "r_cg_rtc.h"
void      R_RTC_Set_CalendarAlarm ( rtc_calendar_alarm_enable_t alarm_enable,
 rtc_calendar_alarm_value_t alarm_val );
```

[Argument(s)]

I/O	Argument	Description
I	rtc_calendar_alarm_enable_t *alarm_enable*;	Comparison flags (year, month, date, day-of-the-week, hour, minute, and second). 0x0 : Comparison proceeds 0x1 : Comparison does not proceed
I	rtc_calendar_alarm_value_t *alarm_val*;	Calendar and time values (year, month, date, day-of-week, time, minute, and second)

Remark 1. The configuration of the comparison flag structure rtc_calendar_alarm_enable_t is shown below.

```
typedef struct {
    uint8_t sec_enb;      /* Second */
    uint8_t min_enb;      /* Minute */
    uint8_t hr_enb;       /* Time */
    uint8_t day_enb;      /* Date */
    uint8_t wk_enb;       /* Day-of-week */
    uint8_t mon_enb;      /* Month */
    uint8_t yr_enb;       /* Year */
} rtc_calendar_alarm_enable_t;
```

Remark 2. The configuration of the calendar and time values rtc_calendar_alarm_value_t is shown below.

```
typedef struct {
    uint8_t rsecar;       /* second */
    uint8_t rminar;       /* Minute */
    uint8_t rhrar;        /* Time */
    uint8_t rdayar;       /* Date */
    uint8_t rwkar;        /* Day-of-week (0: Sunday, 6: Saturday) */
    uint8_t rmonar;       /* Month */
    uint16_t ryrar;       /* Year */
} rtc_calendar_alarm_value_t;
```

[Return value]

None.

R_RTC_Set_BinaryAlarm

Sets the condition for the alarm interrupt (ALM) and allows detection of ALM (binary count mode).

[Syntax]

```
void    R_RTC_Set_BinaryAlarm ( uint32_t alarm_enable, uint32_t alarm_val );
```

[Argument(s)]

I/O	Argument	Description
I	uint32_t *alarm_enable*;	Comparison flag 0x0 : Comparison does not proceed 0x1 : Comparison proceeds
I	uint32_t *alarm_val*;	The value of the binary counter

[Return value]

None.

R_RTC_Set_ConstPeriodInterruptOn

Sets the period of the periodic interrupt (PRD) and allows detection of the PRD.

[Syntax]

```
#include          "r_cg_rtc.h"
void      R_RTC_Set_ConstPeriodInterruptOn ( rtc_int_period_t period );
```

[Argument(s)]

I/O	Argument	Description
I	rtc_int_period_t *period*;	Period of the periodic interrupt (PRD). PES_2_SEC : 2 seconds PES_1_SEC : 1 second PES_1_2_SEC : 1/2 second PES_1_4_SEC : 1/4 second PES_1_8_SEC : 1/8 second PES_1_16_SEC : 1/16 second PES_1_32_SEC : 1/32 second PES_1_64_SEC : 1/64 second PES_1_128_SEC : 1/128 second PES_1_256_SEC : 1/256 second

[Return value]

None.

R_RTC_Set_ConstPeriodInterruptOff

Prohibits detection of the periodic interrupt (PRD).

[Syntax]

```
void    R_RTC_Set_ConstPeriodInterruptOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_CarryInterruptOn

Allows detection of the carry interrupt (CUP).

[Syntax]

```
void    R_RTC_Set_CarryInterruptOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_CarryInterruptOff

Prohibits detection of the carry interrupt (CUP).

[Syntax]

```
void    R_RTC_Set_CarryInterruptOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_RTCOUTOn

Sets the RTCOUT output period and starts RTCOUT output.

[Syntax]

```
#include          "r_cg_rtc.h"
void      R_RTC_Set_RTCOUTOn ( rtc_rtcout_period_t rtcout_freq );
```

[Argument(s)]

I/O	Argument	Description
I	rtc_rtcout_period_t *rtcout_freq*;	RTCOUT output period RTCOUT_1HZ : 1Hz RTCOUT_64HZ : 64Hz

[Return value]

None.

R_RTC_Set_RTCOUTOff

Ends RTCOUT output.

[Syntax]

```
void    R_RTC_Set_RTCOUTOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Start

Starts counting.

[Syntax]

```
void    R_RTC_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Stop

Ends counting.

[Syntax]

```
void    R_RTC_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Restart

Initializes the counter then starts counting.

Remark 1. When the realtime clock is operating in the calendar counting mode, this API function initializes the counters to the values specified by the argument *counter_write_val*.

Remark 2. When the realtime clock is operating in the binary counting mode, this API function ignores the value specified by the argument *counter_write_val* and clears the counter to zero.

[Syntax]

```
#include          "r_cg_rtc.h"
void      R_RTC_Restart ( rtc_calendarcounter_value_t counter_write_val );
```

[Argument(s)]

I/O	Argument	Description
I	rtc_calendarcounter_value_t *counter_write_val*;	Initial value (year, month, date, day-of-week, time, minute, and second)

Remark The configuration of the initial value rtc_calendarcounter_value_t is shown below.

```
typedef struct {
    uint8_t rseccnt;     /* second */
    uint8_t rmincnt;     /* Minute */
    uint8_t rhrcnt;      /* Time */
    uint8_t rdaycnt;     /* Date */
    uint8_t rwkcnt;      /* Day-of-week (0: Sunday, 6: Saturday) */
    uint8_t rmoncnt;     /* Month */
    uint16_t ryrcnt;     /* Year */
} rtc_calendarcounter_value_t;
```

[Return value]

None.

R_RTC_Set_CalendarCounterValue

Sets the calendar and time values.

[Syntax]

```
#include          "r_cg_rtc.h"
void      R_RTC_Set_CalendarCounterValue ( rtc_calendarcounter_value_t counter_write_val );
```

[Argument(s)]

I/O	Argument	Description
I	rtc_calendarcounter_value_t *counter_write_val*;	Calendar and time values (year, month, date, day-of-week, time, minute, and second)

Remark The configuration of the calendar and time values rtc_calendarcounter_value_t is shown below.

```
typedef struct {
    uint8_t rseccnt;      /* second */
    uint8_t rmincnt;      /* Minute */
    uint8_t rhrcnt;       /* Time */
    uint8_t rdaycnt;      /* Date */
    uint8_t rwkcnt;       /* Day-of-week (0: Sunday, 6: Saturday) */
    uint8_t rmoncnt;      /* Month */
    uint16_t ryrcnt;      /* Year */
} rtc_calendarcounter_value_t;
```

[Return value]

None.

R_RTC_Get_CalendarCounterValue

Gets the calendar and time values.

[Syntax]

```
#include          "r_cg_rtc.h"
void      R_RTC_Get_CalendarCounterValue ( rtc_calendarcounter_value_t * const counter_read_val );
```

[Argument(s)]

I/O	Argument	Description
O	rtc_calendarcounter_value_t * const *counter_read_val*;	Pointer to the area where the obtained calendar and time values (year, month, date, day-of-week, time, minute, and second) are to be stored

Remark The configuration of the calendar and time values rtc_calendarcounter_value_t is shown below.

```
typedef struct {
    uint8_t rseccnt;     /* second */
    uint8_t rmincnt;     /* Minute */
    uint8_t rhrcnt;      /* Time */
    uint8_t rdaycnt;     /* Date */
    uint8_t rwkcnt;      /* Day-of-week (0: Sunday, 6: Saturday) */
    uint8_t rmoncnt;     /* Month */
    uint16_t ryrcnt;     /* Year */
} rtc_calendarcounter_value_t;
```

[Return value]

None.

R_RTC_Set_BinaryCounterValue

Sets the value of the binary counter.

[Syntax]

```
void    R_RTC_Set_BinaryCounterValue ( uint32_t counter_write_val );
```

[Argument(s)]

I/O	Argument	Description
I	uint32_t *counter_write_val*;	The value of the binary counter

[Return value]

None.

R_RTC_Get_BinaryCounterValue

Gets the value of the binary count.

[Syntax]

```
void    R_RTC_Get_BinaryCounterValue ( uint32_t * const counter_read_val );
```

[Argument(s)]

I/O	Argument	Description
O	uint32_t * const *counter_read_val*;	Pointer to an area where the obtained value of the binary counter is to be stored

[Return value]

None.

R_RTC_Get_CalendarTimeCaptureValue*n*

Gets the captured calendar time value.

[Syntax]

```
#include          "r_cg_rtc.h"
void      R_RTC_Get_CalendarTimeCaptureValuen ( rtc_calendarcounter_value_t * const
 counter_read_val );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	rtc_calendarcounter_value_t * const *counter_read_val*;	Pointer to the area where the obtained calendar and time values.

Remark The configuration of the calendar and time values rtc_calendarcounter_value_t is shown below.

```
typedef struct {
    uint8_t rseccnt;          /* Second */
    uint8_t rmincnt;          /* Minute */
    uint8_t rhrcnt;           /* Time */
    uint8_t rdaycnt;          /* Date */
    uint8_t rwkcnt;           /* Day-of-week (0: Sunday, 6: Saturday) */
    uint8_t rmoncnt;          /* Month */
    uint16_t ryrcnt;          /* Year */
} rtc_calendarcounter_value_t;
```

[Return value]

None.

R_RTC_Get_BinaryTimeCaptureValue*n*

Gets the value of the binary count.

[Syntax]

void R_RTC_Get_BinaryTimeCaptureValue*n* (uint32_t * const *counter_read_val*);

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	uint32_t * const *counter_read_val*;	The value of the binary counter.

[Return value]

None.

Usage example

Correct leap seconds by alarm interrupt. (Return to 58 seconds at 59 seconds before changing the date.)

[GUI setting example]

Realtime Clock				Used
	RTC			Used
			Realtime clock operation setting	Used
			Counting mode	Calendar
			Hour mode	24-hour mode
			Readtime clock initial value	2000/1/1 23:59
			Enable time capture event input pin	Unused
			Enabel alarm interrupt	Used
			Year	2000
			Month	1
			Date	1
			Day of week	Saturday
			Hour	23
			Minute	59
			Second	59
			Priority (ALM)	Level 15 (highest)
			Enable periodic interrupt (PRD)	Unused
			Enable carry interrupt (CUP)	Unused

[API setting example]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start RTC counter */
    R_RTC_Start();

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_rtc_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile rtc_calendarcounter_value_t counter_val;
/* End user code. Do not edit comment generated here */

static void r_rtc_alm_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Disable ALM interrupt */
    IEN(RTC,ALM) = 0U;

    /* Get RTC calendar counter value */
    R_RTC_Get_CalendarCounterValue((rtc_calendarcounter_value_t *)&counter_val);

    /* Change the seconds */
    counter_val.rseccnt = 0x58U;

    /* Set RTC calendar counter value */
    R_RTC_Set_CalendarCounterValue(counter_val);
    /* End user code. Do not edit comment generated here */
}
```

3.2.23 Watchdog timer (WDT)

Below is a list of API functions output by the Code Generator for watchdog timer use.

Table 3.23 API Functions: [Watchdog Timer]

API Function Name	Function
R_WDT_Create	Performs initialization necessary to control the watchdog timer.
R_WDT_Restart	Clears the watchdog timer counter and resumes counting.
R_WDT_Create_UserInit	Performs user-defined initialization relating to the watchdog timer.
r_wdt_wuni_interrupt	Performs processing in response to the non-maskable/maskable interrupt.
r_wdt_nmi_interrupt	Performs processing in response to the non-maskable interrupt.

R_WDT_Create

Performs initialization necessary to control the watchdog timer.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_WDT_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_WDT_Restart

Clears the watchdog timer counter and resumes counting.

[Syntax]

```
void    R_WDT_Restart ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_WDT_Create_UserInit

Performs user-defined initialization relating to the watchdog timer.

Remark This API function is called as the R_WDT_Create callback routine.

[Syntax]

```
void    R_WDT_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_wdt_wuni_interrupt

Performs processing in response to the non-maskable/maskable interrupt.

Remark This API function is called to run interrupt processing for the non-maskable / maskable interrupt, which is generated when the down-counter underflows or refreshing proceeds.

[Syntax]

```
static void     r_wdt_wuni_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_wdt_nmi_interrupt

Performs processing in response to the non-maskable interrupt.

Remark This API function is called to run interrupt processing for the non-maskable interrupt, which is generated when the down-counter underflows or refreshing proceeds.

[Syntax]

```
static void     r_wdt_nmi_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

Usage example

For each main loop clears the watchdog timer counter and resumes counting.
Perform software reset at the counter underflows.

[GUI setting example]

Watchdog tTimer				Used
	WDT			Used
			Start mode setting	Auto-start mode
			Clock division ratio	PCLK/128
			Frequency	46.875 (kHz)
			Timeout cycle	16384 (cycles)
			Timeout period	349.525 (ms)
			Window position setting (Start position)	100 (%)
			Window position setting (End position)	0 (%)
			Reset interrupt request setting	Interrupt request output
			Priority	Level 15 (highest)

[API setting example]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    while (1U)
    {
        /* Restarts WDT module */
        R_WDT_Restart();
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_wdt_user.c

```
static void r_wdt_wuni_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Software reset */
    SYSTEM.PRCR.WORD = 0xA502;
    SYSTEM.SWRR = 0xA501;
    /* End user code. Do not edit comment generated here */
}
```

3.2.24 Independent watchdog timer (IWDT)

Below is a list of API functions output by the Code Generator for independent watchdog timer use.

Table 3.24 API Functions: [Independent Watchdog Timer]

API Function Name	Function
R_IWDT_Create	Performs initialization necessary to control the independent watchdog timer.
R_IWDT_Create_UserInit	Performs user-defined initialization relating to the independent watchdog timer.
r_iwdt_nmi_interrupt	Performs processing in response to the non-maskable interrupt (WUNI).
r_iwdt_iwuni_interrupt	Performs processing in response to the non-maskable/maskable interrupt.
R_IWDT_Restart	Clears the independent watchdog timer counter and resumes counting.

R_IWDT_Create

Performs initialization necessary to control the independent watchdog timer.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_IWDT_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IWDT_Create_UserInit

Performs user-defined initialization relating to the independent watchdog timer.

Remark This API function is called as the R_IWDT_Create callback routine.

[Syntax]

```
void    R_IWDT_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_iwdt_nmi_interrupt

Performs processing in response to the non-maskable interrupt (WUNI).

Remark This API function is called to run interrupt processing for the non-maskable interrupt (WUNI), which is generated when the down-counter underflows or refreshing proceeds outside the period where it is permitted.

[Syntax]

```
static void     r_iwdt_nmi_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_iwdt_iwuni_interrupt

Performs processing in response to the non-maskable/maskable interrupt.

Remark This API function is called to run interrupt processing for the non-maskable / maskable interrupt, which is generated when the down-counter underflows or refreshing proceeds.

[Syntax]

```
static void     r_iwdt_iwuni_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IWDT_Restart

Clears the independent watchdog timer counter and resumes counting.

[Syntax]

```
void    R_IWDT_Restart ( void );
```

[Argument(s)]

None.

[Return value]

None.

Usage example

For each main loop clears the watchdog timer counter and resumes counting.
Perform software reset at the counter underflows.

[GUI setting example]

Independent Watchdog Timer				Used
	IWDT			Used
			Stert mode setting	Auto-start mode
			Clock division ratio	IWDTCLK
			Frequency	120 (kHz)
			Timeout cycle	16384 (cycles)
			Timeout period	136.533 (ms)
			Window position setting (Start position)	100 (%)
			Window position setting (End position)	0 (%)
			Sleep mode count stop control setting	Enabled
			Reset interrupt request setting	Interrupt request output
			Priority	Level 15 (highest)

[API setting example]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    while (1U)
    {
        /* Restarts IWDT module */
        R_IWDT_Restart();
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_iwdt_user.c

```
static void r_iwdt_iwuni_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Software reset */
    SYSTEM.PRCR.WORD = 0xA502;
    SYSTEM.SWRR = 0xA501;
    /* End user code. Do not edit comment generated here */
}
```

3.2.25 Serial communications interface (SCI)

Below is a list of API functions output by the Code Generator for serial communications interface use.

Table 3.25 API Functions: [Serial Communications Interface]

API Function Name	Function
R_SCIn_Create	Performs initialization necessary to control the serial communications interface.
R_SCIn_Create_UserInit	Performs user-defined initialization related to the serial communications interface.
r_scin_transmitend_interrupt	Performs processing in response to the transmit-end interrupts.
r_scin_transmit_interrupt	Performs processing in response to the transmit-data-empty interrupts.
r_scin_receive_interrupt	Performs processing in response to the receive-data-full interrupts.
r_scin_receiveerror_interrupt	Performs processing in response to the receive error interrupts.
R_SCIn_Start	Starts SCI communication.
R_SCIn_Stop	Ends SCI communication.
R_SCIn_Serial_Send	Starts SCI transmission (synchronous mode).
R_SCIn_Serial_Receive	Starts SCI reception (synchronous mode).
R_SCIn_Serial_Multiprocessor_Send	Starts SCI transmission (multi-processor communications function).
R_SCIn_Serial_Multiprocessor_Receive	Starts SCI reception (multi-processor communications function).
R_SCIn_Serial_Send_Receive	Starts SCI transmission/reception (clock synchronous mode).
R_SCIn_SmartCard_Send	Starts SCI transmission (smart card interface mode).
R_SCIn_SmartCard_Receive	Starts SCI reception (smart card interface mode).
R_SCIn_IIC_Master_Send	Starts SCI master transmission (simple I2C mode).
R_SCIn_IIC_Master_Receive	Starts SCI master reception (simple I2C mode).
R_SCIn_SPI_Master_Send	Starts SCI master transmission (simple SPI mode).
R_SCIn_SPI_Master_Send_Receive	Starts SCI master transmission/reception (simple SPI mode).
R_SCIn_SPI_Slave_Send	Starts SCI slave transmission (simple SPI mode).
R_SCIn_SPI_Slave_Send_Receive	Starts SCI slave transmission/reception (simple SPI mode).
R_SCIn_IIC_StartCondition	Sends the start bit.
R_SCIn_IIC_StopCondition	Sends the stop bit.
r_scin_callback_transmitend	Performs processing in response to the transmit-end interrupts.
r_scin_callback_receiveend	Performs processing in response to the receive-data-full interrupts.
r_scin_callback_receiveerror	Performs processing in response to the receive error interrupts.

R_SCI*n*_Create

Performs initialization necessary to control the serial communication interface.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_SCIn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCI*n*_Create_UserInit

Performs user-defined initialization related to the serial communications interface.

Remark This API function is called as the R_SCIn_Create callback routine.

[Syntax]

```
void R_SCIn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci*n*_transmitend_interrupt

Performs processing in response to the transmit-end interrupts.

Remark This API function is called to run interrupt processing for the transmit-end interrupts.

[Syntax]

```
static void    r_sci n_transmitend_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci*n*_transmit_interrupt

Performs processing in response to the transmit-data-empty interrupts.

Remark This API function is called to run interrupt processing for the transmit-data-empty interrupts.

[Syntax]

```
static void    r_sci n_transmit_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci*n*_receive_interrupt

Performs processing in response to the receive-data-full interrupts.

Remark This function is called to run interrupt processing for the receive-data-full interrupts.

[Syntax]

```
static void    r_sci n_receive_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci*n*_receiveerror_interrupt

Performs processing in response to the receive error interrupts.

Remark This API function is called to run interrupt processing for the receive error interrupts.

[Syntax]

```
static void    r_sci n_receiveerror_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCI*n*_Start

Starts SCI communication.

[Syntax]

```
void    R_SCIn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCI*n*_Stop

Ends SCI communication.

[Syntax]

```
void    R_SCIn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCI*n*_Serial_Send

Starts SCI transmission (asynchronous mode).

Remark 1. This API function repeats the byte-level SCI transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

Remark 2. When performing a SCI transmission, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include        "r_cg_macrodriver.h"
MD_STATUS     R_SCIn_Serial_Send ( uint8_t * const tx_buf, uint16_t tx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint8_t * const *tx_buf*;	Pointer to a buffer storing the transmission data
I	uint16_t *tx_num*;	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

R_SCI*n*_Serial_Receive

Starts SCI reception (asynchronous mode).

Remark 1. This API function repeats SCI reception in byte units the number of times specified by the argument *rx_num* and then stores the received data in the buffer at the location specified by the argument *rx_buf*.

Remark 2. When performing a SCI reception, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include        "r_cg_macrodriver.h"
MD_STATUS     R_SCIn_Serial_Receive ( uint8_t * const rx_buf, uint16_t rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	uint8_t * const *rx_buf*;	Pointer to a buffer to store the reception data
I	uint16_t *rx_num*;	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *rx_num* specification

R_SCI*n*_Serial_Multiprocessor_Send

Starts SCI transmission (multi-processor communications function).

Remark 1. This API function repeats the byte-level SCI transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

Remark 2. When performing a SCI transmission, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include          "r_cg_macrodriver.h"
MD_STATUS     R_SCIn_Serial_Multiprocessor_Send (uint8_t * const id_buf, uint16_t id_num,
 uint8_t * const tx_buf, uint16_t tx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint8_t * const *id_buf*;	Pointer to a buffer storing the transmission ID
I	uint16_t *id_num*;	Total amount of ID to send
I	uint8_t * const *tx_buf*;	Pointer to a buffer storing the transmission data
I	uint16_t *tx_num*;	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

R_SCI*n*_Serial_Multiprocessor_Receive

Starts SCI reception (multi-processor communications function).

Remark 1. This API function repeats SCI reception in byte units the number of times specified by the argument *rx_num* and then stores the received data in the buffer at the location specified by the argument *rx_buf*.

Remark 2. When performing a SCI reception, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include        "r_cg_macrodriver.h"
MD_STATUS    R_SCIn_Serial_Multiprocessor_Receive ( uint8_t * const rx_buf, uint16_t rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	uint8_t * const *rx_buf*;	Pointer to a buffer to store the reception data
I	uint16_t *rx_num*;	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *rx_num* specification

R_SCI*n*_Serial_Send_Receive

Starts SCI transmission/reception (clock synchronous mode).

Remark 1. This API function repeats SCI transmission in byte units the number of times specified by the argument *tx_num* from the buffer at the location specified by the argument *tx_buf*.

Remark 2. This API function repeats SCI reception processing in byte units the number of times specified by the argument *rx_num* and then stores the received data in the buffer at the location specified by the argument *rx_buf*.

Remark 3. When performing a SCI transmission/reception, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include          "r_cg_macrodriver.h"
MD_STATUS     R_SCIn_Serial_Send_Receive ( uint8_t * const tx_buf, uint16_t tx_num,
 uint8_t * const rx_buf, uint16_t rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint8_t * const *tx_buf*;	Pointer to a buffer storing the transmission data
I	uint16_t *tx_num*;	Total amount of data to send
O	uint8_t * const *rx_buf*;	Pointer to a buffer to store the reception data
I	uint16_t *rx_num*;	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

R_SCI*n*_SmartCard_Send

Starts SCI transmission (smart card interface mode).

Remark 1. This API function repeats the byte-level SCI transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

Remark 2. When performing a SCI transmission, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include          "r_cg_macrodriver.h"
MD_STATUS     R_SCIn_SmartCard_Send ( uint8_t * const tx_buf, uint16_t tx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint8_t * const *tx_buf*;	Pointer to a buffer storing the transmission data
I	uint16_t *tx_num*;	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

R_SCI*n*_SmartCard_Receive

Starts SCI reception (smart card interface mode).

Remark 1. This API function repeats SCI reception in byte units the number of times specified by the argument *rx_num* and then stores the received data in the buffer at the location specified by the argument *rx_buf*.

Remark 2. When performing a SCI reception, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include        "r_cg_macrodriver.h"
MD_STATUS     R_SCIn_SmartCard_Receive ( uint8_t * const rx_buf, uint16_t rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	uint8_t * const *rx_buf*;	Pointer to a buffer to store the reception data
I	uint16_t *rx_num*;	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *rx_num* specification

R_SCI*n*_IIC_Master_Send

Starts SCI master transmission (simple I2C mode).

Remark 1. This API function handles SCI master transmission to the slave device at the address specified by the argument *adr* and the R/W#bit. SCI master transmission in byte units is repeated the number of times specified by the argument *tx_num* from the buffer at the location specified by the argument *tx_buf*.

Remark 2. This API function internally calls R_SCIn_IIC_StartCondition to handle processing to start SCI master transmission.

Remark 3. When performing a SCI master transmission, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
void    R_SCIn_IIC_Master_Send ( uint8_t adr, uint8_t * const tx_buf, uint16_t tx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint8_t *adr*;	Slave address
I	uint8_t * const *tx_buf*;	Pointer to a buffer storing the transmission data
I	uint16_t *tx_num*;	Total amount of data to send

[Return value]

None.

R_SCI*n*_IIC_Master_Receive

Starts SCI master reception (simple I2C mode).

Remark 1. This API function handles SCI master transmission to the slave device at the address specified by the argument *adr*. SCI master reception in byte units is repeated the number of times specified by the argument *rx_num* and the received data are stored in the buffer at the location specified by the argument *rx_buf*.

Remark 2. This API function internally calls R_SCIn_IIC_StartCondition to handle processing to start SCI master reception.

Remark 3. When performing a SCI master reception, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
void    R_SCIn_IIC_Master_Receive ( uint8_t adr, uint8_t * const rx_buf, uint16_t rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint8_t *adr*;	Slave address
O	uint8_t * const *rx_buf*;	Pointer to a buffer to store the reception data
I	uint16_t *rx_num*;	Total amount of data to receive

[Return value]

None.

R_SCI*n*_SPI_Master_Send

Starts SCI master transmission (simple SPI mode).

Remark 1. This API function repeats the byte-level SCI master transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

Remark 2. When performing a SCI master transmission, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include          "r_cg_macrodriver.h"
MD_STATUS     R_SCIn_SPI_Master_Send ( uint8_t * const tx_buf, uint16_t tx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint8_t * const *tx_buf*;	Pointer to a buffer storing the transmission data
I	uint16_t *tx_num*;	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

R_SCI*n*_SPI_Master_Send_Receive

Starts SCI master transmission/reception (simple SPI mode).

Remark 1. This API function repeats SCI master transmission in byte units the number of times specified by the argument *tx_num* from the buffer at the location specified by the argument *tx_buf*.

Remark 2. This API function repeats SCI master reception in byte units the number of times specified by the argument *rx_num* and the received data are stored in the buffer at the location specified by the argument *rx_buf*.

Remark 3. When performing a SCI master transmission/reception, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include          "r_cg_macrodriver.h"
MD_STATUS     R_SCIn_SPI_Master_Send_Receive ( uint8_t * const tx_buf, uint16_t tx_num,
uint8_t * const rx_buf, uint16_t rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint8_t * const *tx_buf*;	Pointer to a buffer storing the transmission data
I	uint16_t *tx_num*;	Total amount of data to send
O	uint8_t * const *rx_buf*;	Pointer to a buffer to store the reception data
I	uint16_t *rx_num*;	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

R_SCI*n*_SPI_Slave_Send

Starts SCI slave transmission (simple SPI mode).

Remark 1. This API function repeats the byte-level SCI slave transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

Remark 2. When performing a SCI slave transmission, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include          "r_cg_macrodriver.h"
MD_STATUS     R_SCIn_SPI_Slave_Send (uint8_t * const tx_buf, uint16_t tx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint8_t * const *tx_buf*;	Pointer to a buffer storing the transmission data
I	uint16_t *tx_num*;	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

R_SCI*n*_SPI_Slave_Send_Receive

Starts SCI slave transmission/reception (simple SPI mode).

Remark 1. This API function repeats SCI slave transmission in byte units the number of times specified by the argument *tx_num* from the buffer at the location specified by the argument *tx_buf*.

Remark 2. This API function repeats SCI slave reception in byte units the number of times specified by the argument *rx_num* and the received data are stored in the buffer at the location specified by the argument *rx_buf*.

Remark 3. When performing a SCI slave transmission/reception, R_SCIn_Start must be called before this API function is called.

[Syntax]

```
#include          "r_cg_macrodriver.h"
MD_STATUS     R_SCIn_SPI_Slave_Send_Receive ( uint8_t * const tx_buf, uint16_t tx_num,
uint8_t * const rx_buf, uint16_t rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint8_t * const *tx_buf*;	Pointer to a buffer storing the transmission data
I	uint16_t *tx_num*;	Total amount of data to send
O	uint8_t * const *rx_buf*;	Pointer to a buffer to store the reception data
I	uint16_t *rx_num*;	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

R_SCI*n*_IIC_StartCondition

Sends the start bit.

Remark This API function is called as the internal function of R_SCIn_IIC_Master_Send and R_SCIn_IIC_Master_Receive.

[Syntax]

```
void    R_SCIn_IIC_StartCondition ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCI*n*_IIC_StopCondition

Sends the stop bit.

[Syntax]

```
void    R_SCIn_IIC_StopCondition ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci*n*_callback_transmitend

Performs processing in response to the transmit-end interrupts.

Remark This API function is called as the r_scin_transmitend_interrupt callback routine.

[Syntax]

```
static void    r_sci n _callback_transmitend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci*n*_callback_receiveend

Performs processing in response to the receive-data-full interrupts.

Remark This API function is called as the r_scin_receive_interrupt callback routine.

[Syntax]

```
static void    r_sci n _callback_receiveend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_sci*n*_callback_receiveerror

Performs processing in response to the receive error interrupts.

Remark This API function is called as the r_scin_receiveerror_interrupt callback routine.

[Syntax]

```
static void    r_sci n _callback_receiveerror ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

Usage example

Transmit capital letter 'A' by UART.

[GUI setting example]

Serial Communications Interface				Used
	SCI6			Used
			Function setting	Asynchronous mode (Transmission)
			TXD6	P00
		AsynchronousMode_Transmit6		Used
			Data length setting	8 bits
			Parity setting	None
			Stop bit length setting	1 bit
			Transfer direction setting	LSB-first
			Transfer clock	Internal clock
			Bit rate	9600 (bps)
			Enable modulation duty correction	Unused
			SCK6 pin function	SCK6 is not used
			Hardware flow control setting	None
			Transmit data handling	Data handled in interrupt service routine
			TXI6 Priority	Level 15 (highest)
			TEI6 Priority (Group BL0)	(Please set priority setting in ICU)
			Transmission end	Used

Interrupt Controller Unit				Used
	ICU			Used
		Group		Used
			Group BL0	Used
			Group BL0 priority	Leve; 15 (highest)

[API setting example]

r_cg_main.c

```
/* Start user code for global. Do not edit comment generated here */
extern volatile uint8_t g_sci6_tx_buf;
/* End user code. Do not edit comment generated here */

void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start the SCI6 channel */
    R_SCI6_Start();

    /* Transmit SCI6 data */
    R_SCI6_Serial_Send((uint8_t *)&g_sci6_tx_buf, 1U);

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_sci_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_sci6_tx_buf;
/* End user code. Do not edit comment generated here */

void R_SCI6_Create_UserInit(void)
{
    /* Start user code. Do not edit comment generated here */
    g_sci6_tx_buf = 'A';
    /* End user code. Do not edit comment generated here */
}

static void r_sci6_callback_transmitend(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop the SCI6 channel */
    R_SCI6_Stop();
    /* End user code. Do not edit comment generated here */
}
```

3.2.26 FIFO embedded serial communications interface (SCIFA)

Below is a list of API functions output by the Code Generator for FIFO embedded serial communications interface use.

Table 3.26 API Functions: [FIFO Embedded Serial Communications Interface]

API Function Name	Function
R_SCIFAn_Create	Performs initialization necessary to control the FIFO embedded serial communications interface.
R_SCIFAn_Start	Starts FIFO embedded SCI communication.
R_SCIFAn_Stop	Ends FIFO embedded SCI communication.
R_SCIFAn_Serial_Send	EStarts FIFO embedded SCI transmission (asynchronous mode).
R_SCIFAn_Serial_Receive	Starts FIFO embedded SCI reception (asynchronous mode).
R_SCIFAn_Serial_Send_Receive	Starts FIFO embedded SCI transmission/reception (clock synchronous mode).
R_SCIFAn_Create_UserInit	Performs user-defined initialization relating to the FIFO embedded serial communications interface.
r_scifan_teif_interrupt	Performs processing in response to the transmit-end interrupts.
r_scifan_txif_interrupt	Performs processing in response to the transmit FIFO data empty interrupts.
r_scifan_rxif_interrupt	Performs processing in response to the receive FIFO data full interrupts.
r_scifan_erif_interrupt	Performs processing in response to the framing error or parity error interrupts.
r_scifan_brif_interrupt	Performs processing in response to the break or overrun interrupts.
r_scifan_drif_interrupt	Performs processing in response to the receive data ready interrupts.
r_scifan_callback_transmitend	Performs processing in response to the transmit-end interrupts.
r_scifan_callback_receiveend	Performs processing in response to the receive FIFO data full interrupts.
r_scifan_callback_error	Performs processing in response to the error interrupts.

R_SCIFA*n*_Create

Performs initialization necessary to control the FIFO embedded serial communications interface.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_SCIFAn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCIFA*n*_Start

Starts FIFO embedded SCI communication.

[Syntax]

```
void    R_SCIFAn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCIFA*n*_Stop

Ends FIFO embedded SCI communication.

[Syntax]

```
void    R_SCIFAn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_SCIFA*n*_Serial_Send

Starts FIFO embedded SCI transmission (asynchronous mode).

Remark 1. This API function repeats the byte-level SCI transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

Remark 2. When performing a SCI transmission, R_SCIFAn_Start must be called before this API function is called.

[Syntax]

```
#include          "r_cg_macrodriver.h"
MD_STATUS     R_SCIFAn_Serial_Send ( uint8_t * const tx_buf, uint16_t tx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint8_t * const *tx_buf*;	Pointer to a buffer storing the transmission data
I	uint16_t *tx_num*;	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument

R_SCIFA*n*_Serial_Receive

Starts FIFO embedded SCI reception (asynchronous mode).

Remark 1. This API function repeats the byte-level SCI reception from the buffer specified in argument *rx_buf* the number of times specified in argument *rx_num*.

Remark 2. When performing a SCI reception, R_SCIFAn_Start must be called before this API function is called.

[Syntax]

```
#include          "r_cg_macrodriver.h"
MD_STATUS     R_SCIFAn_Serial_Receive ( uint8_t * const rx_buf, uint16_t rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	uint8_t * const *rx_buf*;	Pointer to a buffer to store the reception data
I	uint16_t *rx_num*;	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument

R_SCIFA*n*_Serial_Send_Receive

Starts FIFO embedded SCI transmission/reception (clock synchronous mode).

Remark 1. This API function repeats the byte-level SCI transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

Remark 2. This API function repeats the byte-level SCI reception from the buffer specified in argument *rx_buf* the number of times specified in argument *rx_num*.

Remark 3. When performing a SCI transmission/reception, R_SCIFAn_Start must be called before this API function is called.

[Syntax]

```
#include          "r_cg_macrodriver.h"
MD_STATUS     R_SCIFAn_Serial_Send_Receive ( uint8_t * const tx_buf, uint16_t tx_num,
 uint8_t * const rx_buf, uint16_t rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint8_t * const *tx_buf*;	Pointer to a buffer storing the transmission data
I	uint16_t *tx_num*;	Total amount of data to send
O	uint8_t * const *rx_buf*;	Pointer to a buffer to store the reception data
I	uint16_t *rx_num*;	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument

R_SCIFA*n*_Create_UserInit

Performs user-defined initialization related to the FIFO embedded serial communications interface.

Remark This API function is called as the R_SCIFAn_Create callback routine.

[Syntax]

```
void    R_SCIFAn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_scifa*n*_teif_interrupt

Performs processing in response to the transmit-end interrupts.

Remark This API function is called to run the interrupt processing for the transmit-end interrupt.

[Syntax]

```
static void     r_scifan_teif_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_scifa*n*_txif_interrupt

Performs processing in response to the transmit FIFO data empty interrupts.

Remark This API function is called to run the interrupt processing for the transmit FIFO data empty interrupt.

[Syntax]

```
static void    r_scifan_txif_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_scifa*n*_rxif_interrupt

Performs processing in response to the receive FIFO data full interrupts.

Remark This API function is called to run the interrupt processing for the receive FIFO data full interrupt.

[Syntax]

```
static void    r_scifan_rxif_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_scifa*n*_erif_interrupt

Performs processing in response to the framing error or parity error interrupts.

Remark This API function is called to run the interrupt processing for the framing error or parity error interrupt.

[Syntax]

```
static void    r_scifan_erif_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_scifa*n*_brif_interrupt

Performs processing in response to the break or overrun interrupts.

Remark This API function is called to run the interrupt processing for the break or overrun interrupt.

[Syntax]

```
static void    r_scifan_brif_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_scifa*n*_drif_interrupt

Performs processing in response to the receive data ready interrupts.

Remark This API function is called to run the interrupt processing for the receive data ready interrupt.

[Syntax]

```
static void     r_scifan_drif_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_scifa*n*_callback_transmitend

Performs processing in response to the transmit-end interrupts.

Remark This API function is called as the r_scifan_teif_interrupt callback routine.

[Syntax]

```
static void    r_scifan_callback_transmitend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_scifa*n*_callback_receiveend

Performs processing in response to the receive FIFO data full interrupts.

Remark This API function is called as the r_scifan_rxif_interrupt callback routine.

[Syntax]

```
static void    r_scifan_callback_receiveend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_scifa*n*_callback_error

Performs processing in response to the error interrupts.

Remark This API function is called as the r_scifan_erif_interrupt or r_scifan_brif_interrupt callback routine.

[Syntax]

```
static void    r_scifan_callback_error ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

Usage example

Transmit capital letter 'A' by UART.

[GUI setting example]

Serial Communications Interface with FIFO				Used
	SCIF9			Used
			Function setting	Asynchronous mode (Transmittion)
			TXD9	PB7
		AsynchronousMode_Transmit9		Used
			Data length setting	8 bits
			Parity setting	None
			Stop bit length setting	1 bit
			Transfer direction setting	LSB-first
			Transfer clock	Internal clock
			Bit rate	9600 (bps)
			Enable modulation duty correction	Unused
			SCK9 pin function	SCK9 is not used
			Enable modem control	Unused
			Transmit FIFO data trigger number	0
			Loop-back test	Disable
			Transmit data handling	Data handled in interrupt service routine
			TXI9 priority	Level 15 (highest)
			TEI9 priority (Group AL0)	(Please set priority setting in ICU)
			Transmission end	Used1

Interrupt Controller Unit				Used
	ICU			Used
		Group		Used
			Group AL0	Used
			Group AL0 priority	Level 15 (highest)

[API setting example]

r_cg_main.c

```
/* Start user code for global. Do not edit comment generated here */
extern volatile uint8_t g_scifa9_tx_buf;
/* End user code. Do not edit comment generated here */

void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start the SCIFA9 channel */
    R_SCIFA9_Start();

    /* Transmit SCIFA9 data */
    R_SCIFA9_Serial_Send((uint8_t *)&g_scifa9_tx_buf, 1U);

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_scifa_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_scifa9_tx_buf;
/* End user code. Do not edit comment generated here */

void R_SCIFA9_Create_UserInit(void)
{
    /* Start user code. Do not edit comment generated here */
    g_scifa9_tx_buf = 'A';
    /* End user code. Do not edit comment generated here */
}

static void r_scifa9_callback_transmitend(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop the SCIFA9 channel */
    R_SCIFA9_Stop();
    /* End user code. Do not edit comment generated here */
}
```

3.2.27 I2C bus interface (RIIC)

Below is a list of API functions output by the Code Generator for I2C bus interface use.

Table 3.27 API Functions: [I2C Bus Interface]

API Function Name	Function
R_RIICn_Create	Performs initialization necessary to control the I2C bus interface.
R_RIICn_Create_UserInit	Performs user-defined initialization relating to the I2C bus interface.
r_riicn_error_interrupt	Performs processing in response to the transfer error/event generation interrupts (EEI).
r_riicn_receive_interrupt	Performs processing in response to the receive data full interrupts (RXI).
r_riicn_transmit_interrupt	Performs processing in response to the transmit data empty interrupts (TXI).
r_riicn_transmitend_interrupt	Performs processing in response to the transmit end interrupts (TEI).
R_RIICn_Start	Starts RIIC communication.
R_RIICn_Stop	Ends RIIC communication.
R_RIICn_Master_Send	Starts RIIC master transmission.
R_RIICn_Master_Receive	Starts RIIC master reception.
R_RIICn_Slave_Send	Starts RIIC slave transmission.
R_RIICn_Slave_Receive	Starts RIIC slave reception.
R_RIICn_StartCondition	Issues the start condition and causes a transfer error and an event generation interrupt (EEI).
R_RIICn_StopCondition	Issues the stop condition and causes a transfer error and an event generation interrupt (EEI).
r_riicn_callback_receiveerror	Of the internal processing for transfer error/event generation interrupts (EEI), this function handles processing specialized in the arbitration-lost detection, NACK detection, and timeout detection.
r_riicn_callback_transmitend	Of the internal processing for transfer error/event generation interrupts (EEI), this function handles processing specialized in the start condition detection in response to calling of R_RIICn_Master_Send.
r_riicn_callback_receiveend	Of the interrupt processing for transfer error/event generation interrupts (EEI), processing specialized in the start condition detection in response to calling of R_RIICn_Master_Receive is performed.

R_RIIC*n*_Create

Performs initialization necessary to control the I2C bus interface.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_RIICn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RIIC*n*_Create_UserInit

Performs user-defined initialization relating to the I2C bus interface.

Remark This API function is called as the R_RIICn_Create callback routine.

[Syntax]

```
void R_RIICn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_riic*n*_error_interrupt

Performs processing in response to the transfer error/event generation interrupts (EEI).

Remark This API function is called to run interrupt processing for the transfer error/event generation interrupts (EEI), which are generated when the I2C bus interface detects the transfer error/event generation (arbi-tration-lost, NACK, timeout, start condition, and stop condition).

[Syntax]

```
static void     r_riicn_error_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_riic*n*_receive_interrupt

Performs processing in response to the receive data full interrupts (RXI).

Remark This API function is called to run interrupt processing for the receive data full interrupts (RXI).

[Syntax]

```
static void    r_riicn_receive_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_riic*n*_transmit_interrupt

Performs processing in response to the transmit data empty interrupts (TXI).

Remark This function is called to run interrupt processing for the transmit data empty interrupts (TXI).

[Syntax]

```
static void    r_riicn_transmit_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_riic*n*_transmitend_interrupt

Performs processing in response to the transmit end interrupts (TEI).

Remark This API function is called to run interrupt processing for the transmit end interrupts (TEI).

[Syntax]

```
static void    r_riicn_transmitend_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RIIC*n*_Start

Starts RIIC communication.

[Syntax]

```
void    R_RIICn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RIIC*n*_Stop

Ends RIIC communication.

[Syntax]

```
void    R_RIICn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RIIC*n*_Master_Send

Starts RIIC master transmission.

Remark 1. This API function handles RIIC master transmission to the slave device at the address specified by the argument *adr* and the R/W#bit. RIIC master transmission in byte units is repeated the number of times specified by the argument *tx_num* from the buffer at the location specified by the argument *tx_buf*.

Remark 2. This API function internally calls R_RIICn_StartCondition to handle processing to start RIIC master transmission.

Remark 3. When performing a RIIC master transmission, R_RIICn_Start must be called before this API function is called.

[Syntax]

```
#include        "r_cg_macrodriver.h"
MD_STATUS     R_RIICn_Master_Send ( uint16_t adr, uint8_t * const tx_buf, uint16_t tx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint16_t *adr*;	Slave address
I	uint8_t * const *tx_buf*;	Pointer to a buffer storing the transmission data
I	uint16_t *tx_num*;	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Bus busy
MD_ERROR2	Invalid argument *adr* specification

R_RIIC*n*_Master_Receive

Starts RIIC master reception.

Remark 1. This API function handles RIIC master transmission to the slave device at the slave address specified by the argument *adr*. RIIC master reception in byte units is repeated the number of times specified by the argument *rx_num* and the received data are stored in the buffer at the location specified by the argument *rx_buf*.

Remark 2. This API function internally calls R_RIICn_StartCondition to handle processing to start RIIC master reception.

Remark 3. When performing a RIIC master reception, R_RIICn_Start must be called before this API function is called.

[Syntax]

```
#include        "r_cg_macrodriver.h"
MD_STATUD    R_RIICn_Master_Receive ( uint16_t adr, uint8_t * const rx_buf, uint16_t rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint16_t *adr*;	Slave address
O	uint8_t * const *rx_buf*;	Pointer to a buffer to store the reception data
I	uint16_t *rx_num*;	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Bus busy
MD_ERROR2	Invalid argument *adr* specification

R_RIIC*n*_Slave_Send

Starts RIIC slave transmission.

Remark 1. This API function repeats the byte-level RIIC slave transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

Remark 2. When performing a RIIC slave transmission, R_RIICn_Start must be called before this API function is called.

[Syntax]

```
#include        "r_cg_macrodriver.h"
MD_STATUS     R_RIICn_Slave_Send ( uint8_t * const tx_buf, uint16_t tx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint8_t * const *tx_buf*;	Pointer to a buffer storing the transmission data
I	uint16_t *tx_num*;	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion

R_RIIC*n*_Slave_Receive

Starts RIIC slave reception.

Remark 1. This API function performs byte-level RIIC slave reception the number of times specified by the argument *rx_num* and stores the data in the buffer specified by the argument *rx_buf*.

Remark 2. When performing a RIIC slave reception, R_RIICn_Start must be called before this API function is called.

[Syntax]

```
#include          "r_cg_macrodriver.h"
MD_STATUS     R_RIICn_Slave_Receive ( uint8_t * const rx_buf, uint16_t rx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	uint8_t * const *rx_buf*;	Pointer to a buffer to store the reception data
I	uint16_t *rx_num*;	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion

R_RIIC*n*_StartCondition

Issues the start condition and causes a transfer error and an event generation interrupt (EEI).

Remark 1. This API function is called as the internal function of R_RIICn_Master_Send and R_RIICn_Master_Receive.

Remark 2. r_riicn_error_interrupt is called in response to calling of this API function.

[Syntax]

```
void    R_RIICn_StartCondition ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RIIC*n*_StopCondition

Issues the stop condition and causes a transfer error and an event generation interrupt (EEI).

Remark r_riicn_error_interrupt is called in response to calling of this API function.

[Syntax]

```
void R_RIICn_StopCondition ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_riic*n*_callback_receiveerror

Of the internal processing for transfer errors and event generation interrupts (EEI), this function handles processing specialized in the arbitration-lost detection, NACK detection, and timeout detection.

Remark This API function is called as the r_riicn_error_interrupt callback routine.

[Syntax]

```
#include          "r_cg_macrodriver.h"
static void       r_riicn_callback_receiveerror ( MD_STATUS status );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	MD_STATUS *status*;	Source of the transfer errors and event generation interrupts MD_ERROR1 : Arbitration-lost detection MD_ERROR2 : Timeout detection MD_ERROR3 : NACK detection

[Return value]

None.

r_riic*n*_callback_transmitend

Of the internal processing for transfer errors and event generation interrupts (EEI), this function handles processing specialized in the start condition detection in response to calling of R_RIICn_Master_Send.

Remark This API function is called as the r_riicn_error_interrupt callback routine.

[Syntax]

```
static void r_riicn_callback_transmitend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_riic*n*_callback_receiveend

Of the internal processing for transfer errors and event generation interrupts (EEI), this function handles processing specialized in the start condition detection in response to calling of R_RIICn_Master_Receive.

Remark This API function is called as the r_riicn_error_interrupt callback routine.

[Syntax]

```
static void r_riicn_callback_receiveend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

Usage example

Master transmission four times

[GUI setting example]

I2C Bus Interface				Used
	RIIC2			Used
			Function setting	I2C mode (Master)
			SCL2	P16
			SDA2	P17
		I2CMaster2		Used
			Baudrate	10 (kbps)
			Enabel noise filter	Used
			Noise filter stage	Single-stage filter
			Enable SDA output delay	Unused
			Enable timeout function	Unused
			Enable master arbitration-lost detection	Used
			Enable NACK transmission arbitration-lost detection	Unused
			Enable transfer suspension during NACK reception	Used
			TXI priority	Level 15 (highest)
			RXI priority	Level 15 (highest)
			TEI2, EEI2 priority (Group BL1)	Level 15 (highest)
			Enable timeout interrupt (TMOI)	Unused
			Enable arbitration-lost interrupt (ALI)	Used
			Enable start condition detection interrupt (STI)	Used
			Enable stop condition detection interrupt (SPI)	Used
			Enable NACK reception interrupt (NAKI)	Used
			Transfer end	Used
			Receive end	Used
			Receive error	Used

Interrupt Controller Unit				Used
	ICU			Used
		Group		Used
			Group BL1	Used
			Group BL1 priority	Level 15 (highest)

[API setting example]

r_cg_main.c

```
/* Start user code for global. Do not edit comment generated here */
extern volatile uint8_t g_riic2_tx_buf[2];
/* End user code. Do not edit comment generated here */

void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start the RIIC2 Bus Interface */
    R_RIIC2_Start();

    /* Send RIIC2 data to slave device */
    R_RIIC2_Master_Send(0x00A0, (uint8_t *)g_riic2_tx_buf, 2U);

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_riic_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_riic2_tx_buf[2];
volatile uint8_t g_riic2_tx_cnt;
/* End user code. Do not edit comment generated here */

void R_RIIC2_Create_UserInit(void)
{
    /* Start user code. Do not edit comment generated here */
    g_riic2_tx_cnt = 0U;
    g_riic2_tx_buf[0] = g_riic2_tx_cnt;
    g_riic2_tx_buf[1] = 0x01;
    /* End user code. Do not edit comment generated here */
}

static void r_riic2_callback_transmitend(void)
{
    /* Start user code. Do not edit comment generated here */
    if ((++g_riic2_tx_cnt) < 4U)
    {
        g_riic2_tx_buf[0] = g_riic2_tx_cnt;
        g_riic2_tx_buf[1] += 0x01;

        /* Send RIIC2 data to slave device */
        R_RIIC2_Master_Send(0x00A0, (uint8_t *)g_riic2_tx_buf, 2U);
    }
    else
    {
        /* Stop the RIIC2 Bus Interface */
        R_RIIC2_Stop();
    }
    /* End user code. Do not edit comment generated here */
}
```

3.2.28 Serial peripheral interface (RSPI)

Below is a list of API functions output by the Code Generator for serial peripheral interface use.

Table 3.28 API Functions: [Serial Peripheral Interface]

API Function Name	Function
R_RSPIn_Create	Performs initialization necessary to control the serial peripheral interface.
R_RSPIn_Create_UserInit	Performs user-defined initialization relating to the serial peripheral interface.
r_rspin_receive_interrupt	Performs processing in response to the receive buffer full interrupts.
r_rspin_transmit_interrupt	Performs processing in response to the transmit buffer error interrupts.
r_rspin_error_interrupt	Performs processing in response to the RSPI error interrupts.
r_rspin_idle_interrupt	Performs processing in response to the RSPI idle interrupts.
R_RSPIn_Start	Starts RSPI communication.
R_RSPIn_Stop	Ends RSPI communication.
R_RSPIn_Send	Starts RSPI transmission.
R_RSPIn_Send_Receive	Starts RSPI transmission/reception.
r_rspin_callback_receiveend	Performs processing in response to the receive buffer full interrupts.
r_rspin_callback_error	Performs processing in response to the RSPI error interrupts.
r_rspin_callback_transmitend	Performs processing in response to the RSPI idle interrupts.

R_RSPI*n*_Create

Performs initialization necessary to control the serial peripheral interface.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_RSPIn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RSPI*n*_Create_UserInit

Performs user-defined initialization relating to the serial peripheral interface.

Remark This API function is called as the R_RSPIn_Create callback routine.

[Syntax]

```
void    R_RSPIn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_rspi*n*_receive_interrupt

Performs processing in response to the receive buffer full interrupts.

Remark This API function is called to run interrupt processing for the receive buffer full interrupt.

[Syntax]

```
static void    r_rspin_receive_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_rspi*n*_transmit_interrupt

Performs processing in response to the transmit buffer empty interrupts.

Remark This API function is called to run interrupt processing for the transmit buffer empty interrupts.

[Syntax]

```
static void    r_rspin_transmit_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_rspi*n*_error_interrupt

Performs processing in response to the RSPI error interrupts.

Remark This API function is called to run interrupt processing for the RSPI error interrupts.

[Syntax]

```
static void    r_rspin_error_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_rspi*n*_idle_interrupt

Performs processing in response to the RSPI idle interrupts.

Remark This API function is called to run interrupt processing for the RSPI idle interrupts.

[Syntax]

```
static void    r_rspin_idle_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RSPI*n*_Start

Starts RSPI communication.

[Syntax]

```
void    R_RSPIn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RSPI*n*_Stop

Ends RSPI communication.

[Syntax]

```
void    R_RSPIn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_RSPI*n*_Send

Starts RSPI transmission.

Remark 1. This API function repeats the byte-level RSPI transmission from the buffer specified in argument *tx_buf* the number of times specified in argument *tx_num*.

Remark 2. When performing a RSPI transmission, R_RSPIn_Start must be called before this API function is called.

[Syntax]

```
#include          "r_cg_macrodriver.h"
MD_STATUS     R_RSPIn_Send ( uint32_t * const tx_buf, uint16_t tx_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint32_t * const *tx_buf*;	Pointer to a buffer storing the transmission data
I	uint16_t *tx_num*;	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

R_RSPI*n*_Send_Receive

Starts RSPI transmission/reception.

Remark 1. This API function repeats RSPI transmission in byte units the number of times specified by the argument *tx_num* from the buffer at the location specified by the argument *tx_buf*.

Remark 2. This API function repeats RSPI reception processing in byte units the number of times specified by the argument *tx_num* and then stores the received data in the buffer at the location specified by the argument *rx_buf*.

Remark 3. When performing a RSPI transmission/reception, R_RSPIn_Start must be called before this API function is called.

[Syntax]

```
#include          "r_cg_macrodriver.h"
MD_STATUS     R_RSPIn_Send_Receive ( uint32_t * const tx_buf, uint16_t tx_num,
 uint32_t * const rx_buf );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint32_t * const *tx_buf*;	Pointer to a buffer storing the transmission data
I	uint16_t *tx_num*;	Total amount of data to send/receive
O	uint32_t * const *rx_buf*;	Pointer to a buffer to store the reception data

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument *tx_num* specification

r_rspi*n*_callback_receiveend

Performs processing in response to the receive buffer full interrupts.

Remark This API function is called as the r_rspin_receive_interrupt callback routine.

[Syntax]

```
static void    r_rspin_callback_receiveend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_rspi*n*_callback_error

Performs processing in response to the RSPI error interrupts.

Remark This API function is called as the r_rspin_error_interrupt callback routine.

[Syntax]

static void r_rspi*n*_callback_error (uint8_t *err_type*);

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	uint8_t *err_type*;	Source of the RSPI error interrupt (x is undefined) *xxxx*00*x*1B : Overrun error detection *xxxx*01*x*0B : Mode fault error detection *xxxx*10*x*0B : Parity error detection

[Return value]

None.

r_rspi*n*_callback_transmitend

Performs processing in response to the RSPI idle interrupts.

Remark. This API function is called as the r_rspin_idle_interrupt callback routine.

[Syntax]

```
static void    r_rspin_callback_transmitend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

Usage example

When the reception of the same number of data as the transmission is completed, the communication is end.

[GUI setting example]

Serial Peripheral Interface				Used
	RSPI0			Used
			Function setting	SPI operation（four-wire method） Master transmit/receive
			RSPCKA pin	PC5 (A)
			MOSIA pin	PC6 (A)
			MISOA pin	PC7 (A)
		Rspi4WireMethod_TransmitReceive_Master0		Used
			Buffer size	64 bits (buffer accessed in words)
			Parity bit	Does not add the parity bit to transmit data and does not check the parity bit of receive data
			Base bit rate	1000(kbps)(Actual value: 1000 Error: 0%)
			Period from beginning of SSL signal assertion to RSPCK oscillation (RSPCK delay）	1 RSPCK
			Period from transmission of final RSPCK edge to negation of the SSL signal (SSL negation delay）	1 RSPCK
			SSL signal non-active period after termination of a serial transfer (next-access delay)	1 RSPCK + 2 PCLK
			MOSI idle value (MOSI output during SSL negation period in master mode)	Final data from previous transfer
			SSLA0 pin	Unused
			SSLA1 pin	PC0 (A) (Active low)
			SSLA2 pin	Unused
			SSLA3 pin	Unused
			Ouptu pin mode selection (RSPCK, SSL and MOSI on master mode / MISO on slave mode)	CMOS output
			Loopback mode selection	Normal mode
			Transmit data handling	Data handled in interrupt service routine0
			SPTI0 priority	Level 15 (highest)
			SPRI0 priority	Level 15 (highest)
			Enable error interrupt (SPEI0)	Used
			SPEI0, SPII0 priority (Group AL0)	Level 15 (highest)
			Transmission end	Used
			Reception end	Used
			Error detection	Used

			Number of commands, number of frames	Number of commands: 1, number of transfer frames: 1
			Command0;	
			Data length	8 bits
			Format	MSB-first
			RSPCK phase	Data variation on odd edge, data samping on even edge
			RSPCK polarity	Low when idle
			Bit rate selection	Base bit rate / 8
			SSL signal assertion	SSL0
			SSL negation operation	Negates all SSL signals upon completion of transfer
			RSPCK delay	1RSPCK
			SSL negation delay	1RSPCK
			Next-access delay	1RSPCK + 2CLK

[API setting example]

r_cg_main.c

```
/* Start user code for global. Do not edit comment generated here */
extern volatile uint32_t g_rspi0_tx_buf[4];
extern volatile uint32_t g_rspi0_rx_buf[4];
/* End user code. Do not edit comment generated here */

void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start the RSPI0 module operation */
    R_RSPI0_Start();

    /* Send and receive RSPI0 data */
    R_RSPI0_Send_Receive((uint32_t *)g_rspi0_tx_buf, 4U, (uint32_t *)g_rspi0_rx_buf);

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_rspi_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint32_t g_rspi0_tx_buf[4];
volatile uint32_t g_rspi0_rx_buf[4];
/* End user code. Do not edit comment generated here */

void R_RSPI0_Create_UserInit(void)
{
    /* Start user code. Do not edit comment generated here */
    g_rspi0_tx_buf[0] = 0x000000FF;
    g_rspi0_tx_buf[1] = 0x0000FF00;
    g_rspi0_tx_buf[2] = 0x00FF0000;
    g_rspi0_tx_buf[3] = 0xFF000000;
    /* End user code. Do not edit comment generated here */
}

static void r_rspi0_callback_receiveend(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop the RSPI0 module operation */
    R_RSPI0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

3.2.29 CRC calculator (CRC)

Below is a list of API functions output by the Code Generator for CRC calculator use.

Table 3.29 CRC API Functions: [CRC calculator]

API Function Name	Function
R_CRC_SetCRC8	Initializes the CRC calculator for 8-bit CRC calculation (CRC generating polynomial: $X^8 + X^2 + X + 1$).
R_CRC_SetCRC16	Initializes the CRC calculator for 16-bit CRC calculation (CRC generating polynomial: $X^{16} + X^{15} + X^2 + 1$).
R_CRC_SetCCITT	Initializes the CRC calculator for 16-bit CRC calculation (CRC generating polynomial: $X^{16} + X^{12} + X^5 + 1$).
R_CRC_SetCRC32	Initializes the CRC calculator for 32-bit CRC calculation (CRC generating polynomial: $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$)
R_CRC_SetCRC32C	Initializes the CRC calculator for 32-bit CRC calculation (CRC generating polynomial: $X^{32} + X^{28} + X^{27} + X^{26} + X^{25} + X^{23} + X^{22} + X^{20} + X^{19} + X^{18} + X^{14} + X^{13} + X^{11} + X^{10} + X^9 + X^8 + X^6 + 1$)
R_CRC_Input_Data	Sets the initial value of the data from which the CRC is to be calculated.
R_CRC_Get_Result	Gets the result of operation.

R_CRC_SetCRC8

Initializes the CRC calculator for 8-bit CRC calculation (CRC generating polynomial: $X^8 + X^2 + X + 1$).

[Syntax]

RX65N/RX651

```
void    R_CRC_SetCRC8 ( void );
```

Other devices

```
#include        "r_cg_crc.h"
void    R_CRC_SetCRC8 ( crc_bitorder order );
```

[Argument(s)]

I/O	Argument	Description
I	crc_bitorder *order*;	CRC calculation switching type CRC_LSB : LSB-first CRC_MSB : MSB-first

[Return value]

None.

R_CRC_SetCRC16

Initializes the CRC calculator for 16-bit CRC calculation (CRC generating polynomial: $X^{16} + X^{15} + X^2 + 1$).

[Syntax]

RX65N/RX651

```
void    R_CRC_SetCRC16 ( void );
```

Other devices

```
#include        "r_cg_crc.h"
void    R_CRC_SetCRC16 ( crc_bitorder order );
```

[Argument(s)]

I/O	Argument	Description
I	crc_bitorder *order*;	CRC calculation switching type CRC_LSB : LSB-first CRC_MSB : MSB-first

[Return value]

None.

R_CRC_SetCCITT

Initializes the CRC calculator for the 16-bit CRC calculation (CRC generating polynomial: $X^{16} + X^{12} + X^5 + 1$).

[Syntax]

RX65N/RX651

```
void    R_CRC_SetCCITT ( void );
```

Other devices

```
#include        "r_cg_crc.h"
void    R_CRC_SetCCITT ( crc_bitorder order );
```

[Argument(s)]

I/O	Argument	Description
I	crc_bitorder *order*;	CRC calculation switching type CRC_LSB : LSB-first CRC_MSB : MSB-first

[Return value]

None.

R_CRC_SetCRC32

Initializes the CRC calculator for the 32-bit CRC calculation (CRC generating polynomial: $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X + 1$).

[Syntax]

```
void    R_CRC_SetCRC32 ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CRC_SetCRC32C

Initializes the CRC calculator for the 32-bit CRC calculation (CRC generating polynomial: $X^{32} + X^{28} + X^{27} + X^{26} + X^{25} + X^{23} + X^{22} + X^{20} + X^{19} + X^{18} + X^{14} + X^{13} + X^{11} + X^{10} + X^{9} + X^{8} + X^{6} + 1$).

[Syntax]

```
void    R_CRC_SetCRC32C ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CRC_Input_Data

Sets the initial value of the data from which the CRC is to be calculated.

[Syntax]

```
void    R_CRC Input_Data ( uint8_t data );
```

```
void    R_CRC Input_Data ( uint32_t data );
```

[Argument(s)]

I/O	Argument	Description
I	uint8_t *data*;	The initial value of the data from which the CRC is to be calculated

I/O	Argument	Description
I	uint32_t *data*;	The initial value of the data from which the CRC is to be calculated

Remark Argument size differs for each device group.

[Return value]

None.

R_CRC_Get_Result

Gets the result of operation.

[Syntax]

```
void    R_CRC_Get_Result ( uint16_t * const result );
```

```
void    R_CRC_Get_Result ( uint32_t * const result );
```

[Argument(s)]

I/O	Argument	Description
O	uint16_t * const *result*;	Pointer to the location where the result of operation is stored

I/O	Argument	Description
O	uint32_t * const *result*;	Pointer to the location where the result of operation is stored

Remark Argument size differs for each device group.

[Return value]

None.

Usage example

Generates CRC code and adds it to transmission data.

A CRC code is generated from the analyzed received data, and it is checked whether the received data is correct.

[GUI setting example]

CRC Calculator				Used
	CRC			Used
			Generate functions for CRC-8 calculation	Used
			Generate functions for CRC-16 calculation	Unused
			Generate functions for CRC-CCITT calculation	Unused
			Initial value	0x0000
			Invert result of calculated value	Unused

[API setting example]

tx_func.c

```
/* Start user code for adding. Do not edit comment generated here */
volatile uint8_t tx_buf[2];
volatile uint16_t result;

void tx_func(void)
{
    /* Set CRC module using CRC8 algorithm */
    R_CRC_SetCRC8(CRC_LSB);

    /* Restore transmit data */
    tx_buf[0] = 0xF0;

    /* Write data to CRC input register */
    R_CRC_Input_Data(tx_buf[0]);

    /* Get result from CRC output register */
    R_CRC_Get_Result((uint16_t *)&result);

    /* Restore CRC code */
    tx_buf[1] = (uint8_t)(result);

    /*** Transmit "tx_buf" ***/
}
/* End user code. Do not edit comment generated here */
```

rx_func.c

```
/* Start user code for adding. Do not edit comment generated here */
volatile uint8_t rx_buf[2];
volatile uint16_t result;
volatile uint8_t err_f;

void rx_func(void)
{
	/* Clare error flag */
	err_f = 0U;

	/*** Receive (Restore the receive data in "rx_buf") ***/

	/* Set CRC module using CRC8 algorithm */
	R_CRC_SetCRC8(CRC_LSB);

	/* Write data to CRC input register */
	R_CRC_Input_Data(rx_buf[0]);

	/* Get result from CRC output register */
	R_CRC_Get_Result((uint16_t *)&result);

	/* Check the receive data */
	if (rx_buf[1] != (uint8_t)(result))
	{
		/* Set error flag */
		err_f = 1U;
	}
}
/* End user code. Do not edit comment generated here */
```

3.2.30 12-bit A/D converter (S12AD)

Below is a list of API functions output by the Code Generator for 12-bit A/D converter use.

Table 3.30 API Functions: [12-Bit A/D Converter]

API Function Name	Function
R_S12ADn_Create	Performs initialization necessary to control the 12-bit A/D converter.
R_S12ADn_Create_UserInit	Performs user-defined initialization relating to the 12-bit A/D converter.
r_s12adn_interrupt	Performs processing in response to the A/D scan end interrupt.
r_s12adn_groupb_interrupt	Performs processing in response to the group B scan end interrupt.
R_S12ADn_Start	Starts A/D conversion.
R_S12ADn_Stop	Ends A/D conversion.
R_S12ADn_Get_ValueResult	Gets the result of conversion.
R_S12ADn_Set_CompareValue	Sets compare level.
r_s12adn_compare_interrupt	Performs processing in response to the compare interrupt.

R_S12AD*n*_Create

Performs initialization necessary to control the 12-bit A/D converter.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_S12ADn_Create ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_S12AD*n*_Create_UserInit

Performs user-defined initialization relating to the 12-bit A/D converter.

Remark This API function is called as the R_S12ADn_Create callback routine.

[Syntax]

```
void R_S12AD n_Create_UserInit ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_s12ad*n*_interrupt

Performs processing in response to the A/D scan end interrupt.

Remark This API function is called to run interrupt processing for the A/D scan end interrupt, which is generated on completion of scanning of the analog inputs.

[Syntax]

```
static void    r_s12adn_interrupt ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_s12ad*n*_groupb_interrupt

Performs processing in response to the group B scan end interrupt.

Remark This function is called to run interrupt processing for the group B scan end interrupt, which is generated when scanning of the analog inputs allocated to group B is completed.

[Syntax]

```
static void    r_s12adn_groupb_interrupt ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_S12AD*n*_Start

Starts A/D conversion.

[Syntax]

```
void    R_S12AD n_Start ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_S12AD*n*_Stop

Ends A/D conversion.

[Syntax]

```
void    R_S12AD n _Stop ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_S12AD*n*_Get_ValueResult

Gets the result of conversion.

[Syntax]

```
#include          "r_cg_s12ad.h"
void      R_S12ADn_Get_ValueResult ( ad_channel_t channel, uint16_t * const buffer );
```

Remark *n* is the unit number.

[Argument(s)]

I/O	Argument	Description
I	ad_channel_t *channel*;	Channel number ADCHANNEL0 : Input channel AN000 ADCHANNEL1 : Input channel AN001 ADCHANNEL2 : Input channel AN002 ADCHANNEL3 : Input channel AN003 ADCHANNEL4 : Input channel AN004 ADCHANNEL5 : Input channel AN005 ADCHANNEL6 : Input channel AN006 ADCHANNEL7 : Input channel AN007 ADCHANNEL8 : Input channel AN008 ADCHANNEL9 : Input channel AN009 ADCHANNEL10 : Input channel AN010 ADCHANNEL11 : Input channel AN011 ADCHANNEL12 : Input channel AN012 ADCHANNEL13 : Input channel AN013 ADCHANNEL14 : Input channel AN014 ADCHANNEL15 : Input channel AN015 ADCHANNEL16 : Input channel AN016 ADCHANNEL17 : Input channel AN017 ADCHANNEL18 : Input channel AN018 ADCHANNEL19 : Input channel AN019 ADCHANNEL20 : Input channel AN020 ADTEMPSENSOR : Extended analog input (temperature sensor output) ADINTERREFVOLT : Extended analog input (internal reference voltage)
O	uint16_t * const *buffer*;	Pointer to the area where the results of conversion are stored

[Return value]

None.

R_S12AD*n*_Set_CompareValue

Sets compare level.

[Syntax]

```
void    R_S12AD n _Set_CompareValue ( uint16_t reg_value0, uint16_t reg_value1);
```

Remark *n* is the unit number.

[Argument(s)]

I/O	Argument	Description
I	uint16_t *reg_value0*;	Register value set to the compare revel register 0
I	uint16_t *reg_value1*;	Register value set to the compare revel register 1

[Return value]

None.

r_s12ad*n*_compare_interrupt

Performs processing in response to the compare interrupt.

Remark This API function is called to run the interrupt processing for the compare interrupt.

[Syntax]

```
static void r_s12adn_compare_interrupt ( void );
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

Usage example

Get the A/D conversion result.

[GUI setting example]

12-Bit A/D Converer				Used
	S12AD0			Used
		AnalogInputChannelMode0		Used
			S12AD0 operation setting	Used
			Operation mode setting	Continuous scan mode
			Self diagnosis setting	Unused
			Disconnection detectionassist setting	Unused
			A / D conversion value count setting	Addition mode
			Analog input channel setting	
			AN000 Convert (Group A)	Used
			AN000 Add/Average AD value	Unused
			AN000Dedicated sample and hold	Unused
			Conversion start trigger (Group A)	Software trigger
			Data placement	Right-alignment
			Automatic clearing	Disable automatic clearing
			Data accuracy	12-bit accuracy
			AN000 Input sumpling time	3.667(us)
			Total conversion time (Group A)	7.167(us)
			Enable AD conversion end interrupt (S12ADI0)	Used
			S12ADI0 priority	Level 15 (highest)
			Window function setting	Unused
			Initial reference data 0 for comparison	0
			Use comparator for AN000	Unused

[API setting example]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start the AD0 converter */
    R_S12AD0_Start();

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_s12ad_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint16_t g_s12ad0_ch000_value;
/* End user code. Do not edit comment generated here */

static void r_s12ad0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Get result from the AD0 channel 0 (AN000) converter */
    R_S12AD0_Get_ValueResult(ADCHANNEL0, (uint16_t *)&g_s12ad0_ch000_value);
    /* End user code. Do not edit comment generated here */
}
```

3.2.31 D/A converter (DA)

Below is a list of API functions output by the Code Generator for D/A converter use.

Table 3.31 API Functions: [D/A Converter]

API Function Name	Function
R_DA_Create	Performs initialization necessary to control the D/A converter.
R_DA_Create_UserInit	Performs user-defined initialization relating to the D/A converter.
R_DAm_Start	Starts D/A conversion.
R_DAm_Stop	Ends D/A conversion.
R_DAm_Set_ConversionValue	Sets the data for D/A conversion.

R_DA_Create

Performs initialization necessary to control the D/A converter.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_DA_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DA_Create_UserInit

Performs user-defined initialization relating to the D/A converter.

Remark This API function is called as the R_DA_Create callback routine.

[Syntax]

```
void    R_DA_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DA*m*_Start

Starts D/A conversion.

[Syntax]

```
void    R_DAm_Start ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DA*m*_Stop

Ends D/A conversion.

[Syntax]

```
void    R_DAm_Stop ( void );
```

Remark *m* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DA*m*_Set_ConversionValue

Sets the data for D/A conversion.

[Syntax]

void R_DA*m*_Set_ConversionValue (uint16_t *reg_value*);

Remark *m* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint16_t *reg_value*;	Data for D/A conversion

[Return value]

None.

Usage example

Start D / A conversion of channels 0 and 1.

[GUI setting example]

D/A Converter				Used
	DA			Used
			D/A converter operation setting	Used
			Use DA0	Used
			Use DA1	Used
			Data format	Right-alignment
			D/A-A/D synchronous setting	Unused

[API setting example]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Set the DA0 converter value */
    R_DA0_Set_ConversionValue(0100U);

    /* Set the DA1 converter value */
    R_DA1_Set_ConversionValue(0200U);

    /* Enable the DA0 converter */
    R_DA0_Start();

    /* Enable the DA1 converter */
    R_DA1_Start();

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

3.2.32 12-bit D/A converter (R12DA)

Below is a list of API functions output by the Code Generator for 12-bit D/A converter use.

Table 3.32 API Functions: [12-Bit D/A Converter]

API Function Name	Function
R_R12DA_Create	Performs initialization necessary to control the 12-bit D/A converter.
R_R12DAn_Start	Starts D/A conversion.
R_R12DAn_Stop	Ends D/A conversion.
R_R12DAn_Set_ConversionValue	Sets the data for D/A conversion.
R_R12DA_Sync_Start	Starts synchronous D/A conversion.
R_R12DA_Sync_Stop	Ends synchronous D/A conversion.
R_R12DA_Create_UserInit	Performs user-defined initialization relating to the 12-bit D/A converter.

R_R12DA_Create

Performs initialization necessary to control the 12-bit D/A converter.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_R12DA_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_R12DA*n*_Start

Starts 12-bit D/A conversion.

[Syntax]

```
void    R_R12DAn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_R12DA*n*_Stop

Ends 12-bit D/A conversion.

[Syntax]

```
void    R_R12DAn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_R12DA*n*_Set_ConversionValue

Sets the data for 12-bit D/A conversion.

[Syntax]

```
void    R_R12DAn_Set_ConversionValue ( uint16_t reg_value );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint16_t *reg_value*;	Data for 12-bit D/A conversion

[Return value]

None.

R_R12DA_Sync_Start

Starts synchronous 12-bit D/A conversion.

[Syntax]

```
void    R_R12DA_Sync_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_R12DA_Sync_Stop

Ends synchronous 12-bit D/A conversion.

[Syntax]

```
void    R_R12DA_Sync_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_R12DA_Create_UserInit

Performs user-defined initialization relating to the 12-bit D/A converter.

Remark This API function is called as the R_R12DA_Create callback routine.

[Syntax]

```
void    R_R12DA_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

Usage example

Start D / A conversion of channels 0 and 1.

[GUI setting example]

D/A Converter				Used
	DA			Used
			D/A converter operation setting	Used
			Use DA0	Used
			Use DA1	Used
			Data format	Right-alignment
			D/A-A/D synchronous setting	Unused

[API setting example]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Set the DA0 converter value */
    R_DA0_Set_ConversionValue(0100U);

    /* Set the DA1 converter value */
    R_DA1_Set_ConversionValue(0200U);

    /* Enable the DA0 converter */
    R_DA0_Start();

    /* Enable the DA1 converter */
    R_DA1_Start();

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

3.2.33 Comparator B (CMPB)

Below is a list of API functions output by the Code Generator for Comparator B use.

Table 3.33 API Functions: [Comparator B]

API Function Name	Function
R_CMPB_Create	Performs initialization necessary to control the Comparator B.
R_CMPB_Create_UserInit	Performs user-defined initialization relating to the Comparator B.
r_cmpb_cmpbn_interrupt	Performs processing in response to the comparator B interrupt.
R_CMPBn_Start	Starts comparison for analog input voltage.
R_CMPBn_Stop	Ends comparison for analog input voltage.

R_CMPB_Create

Performs initialization necessary to control the Comparator B.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_CMPB_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CMPB_Create_UserInit

Performs user-defined initialization relating to the Comparator B.

Remark This API function is called as the R_CMPB_Create callback routine.

[Syntax]

```
void    R_CMPB_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_cmpb_cmpb*n*_interrupt

Performs processing in response to the comparator B interrupt.

Remark This API function is called to run interrupt processing for the comparator B*n* interrupt, which is generated when the comparison result changes at this time.

[Syntax]

```
static void    r_cmpb_cmpbn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CMPB*n*_Start

Starts comparison for analog input voltage.

[Syntax]

```
void    R_CMPBn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CMPB*n*_Stop

Ends comparison for analog input voltage.

[Syntax]

```
void    R_CMPBn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

Usage example

Set the flag to notify changing the comparison result.

[GUI setting example]

Comparator B				Used
	CMPB			Used
			Comparator B0	Used
			Comparator B1	Unused
			Comparator B2	Unused
			Comparator B3	Unused
			Comparator B0, B1 speed setting	Low speed
			Comparator B2, B3 speed setting	Low speed
			Comparator B0 Mode setting	Normal
			Comparator B0 Referance voltage setting	CVREFB0 input
			Comparator b0 Enable digital filter	Unused
			Comparator B0 Enable output (CMPOB0)	Unused
			Enable comparator B0 interrupt (CMPB0)	Used
			Comparator B0 Edge select	Rising
			Comparator B0 priority	Level 15 (highest)

[API setting example]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start comparator B0 */
    R_CMPB0_Start();

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_cmpb_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_cmpb0_f;
/* End user code. Do not edit comment generated here */

void R_CMPB_Create_UserInit(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Clear the flag */
    g_cmpb0_f = 0U;
    /* End user code. Do not edit comment generated here */
}

static void r_cmpb_cmpb0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Set the flag */
    g_cmpb0_f = 1U;
    /* End user code. Do not edit comment generated here */
}
```

3.2.34 Data operation circuit (DOC)

Below is a list of API functions output by the Code Generator for data operation circuit.

Table 3.34 API Functions: [Data Operation Circuit]

API Function Name	Function
R_DOC_Create	Performs initialization necessary to control the data operation circuit.
R_DOC_Create_UserInit	Performs user-defined initialization relating to the data operation circuit.
r_doc_dopcf_interrupt	Performs processing in response to the data operation circuit interrupt.
R_DOC_SetMode	Sets the operating mode and the initial value of the reference value for use by the data operation circuit.
R_DOC_WriteData	Sets the input value (value for comparison with, addition to, or subtraction from the reference value) for use by the data operation circuit.
R_DOC_GetResult	Gets the result of operation.
R_DOC_ClearFlag	Clears the data operation circuit flag.

R_DOC_Create

Performs initialization necessary to control the data operation circuit.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_DOC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DOC_Create_UserInit

Performs user-defined initialization relating to the data operation circuit.

Remark This API function is called as the R_DOC_Create callback routine.

[Syntax]

```
void    R_DOC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_doc_dopcf_interrupt

Performs processing in response to the data operation circuit interrupt.

Remark This API function is called to run interrupt processing for the data operation circuit interrupt, which is generated when the result of data comparison satisfies the condition for detection, the result of addition is greater than 0xFFFF, or the result of subtraction is less than 0x00.

[Syntax]

```
static void     r_doc_dopcf_interrupt ( void );
```

Remark API function name differs for each device group.

[Argument(s)]

None.

[Return value]

None.

R_DOC_SetMode

Sets the operating mode and the initial value of the reference value for use by the data operation circuit.

Remark 1. When COMPARE_MISMATCH or COMPARE_MATCH (data comparison mode) is specified as the mode of operation, the 16-bit reference value is stored in the DOC data setting register (DODSR).

Remark 2. When ADDITION (data addition mode) or SUBTRACTION (data subtraction mode) is specified for the mode (operation mode), the 16-bit value is stored in the DOC data setting register (DODSR) as the initial value.

[Syntax]

```
#include          "r_cg_doc.h"
void       R_DOC_SetMode ( doc_mode_t mode, uint16_t value );
```

[Argument(s)]

I/O	Argument	Description
I	doc_mode_t *mode*;	Operating modes (including the condition for detection) COMPARE_MISMATCH : Data comparison mode (mismatch) COMPARE_MATCH : Data comparison mode (match) ADDITION : Data addition mode SUBTRACTION : Data subtraction mode
I	uint16_t *value*;	Initial value of the reference value for use by the DOC

[Return value]

None.

R_DOC_WriteData

Sets the value for comparison with, addition to, or subtraction from the reference value.

[Syntax]

```
void    R_DOC_WriteData ( uint16_t data );
```

[Argument(s)]

I/O	Argument	Description
I	uint16_t *data*;	Input data for use in operation

[Return value]

None.

R_DOC_GetResult

Gets the result of operation.

[Syntax]

```
void    R_DOC_GetResult ( uint16_t * const data );
```

[Argument(s)]

I/O	Argument	Description
O	uint16_t * const *data*;	Pointer to the location where the result of operation is to be stored

[Return value]

None.

R_DOC_ClearFlag

Clears the data operation circuit flag.

[Syntax]

```
void    R_DOC_ClearFlag ( void );
```

[Argument(s)]

None.

[Return value]

None.

Usage example

Adds the array data and when it becomes larger than "FFFFh" gets the addition result by interrupt.(Data addition mode)

Subsequently, the mode is changed to the Data comparison mode (mismatch), and an interrupt is generated when anything other than "000h" is detected in the array data.

[GUI settings]

Data Operation Circuit				Used
	DOC			Used
			Generate function for settingDOC operation mode	Used
			Operation mode	Data addition mode
			Comparision reference/ Initial value of addition or subtraction result	0
			Enable data operation circuit interrupt(DOPCF)	Used
			DOPCF Priority (Group BL0)	Level 15 (highest)

Interrupt Controller Unit				Used
	ICU			Used
		Group		Used
			Group BL0	Used
			Group BL0 Priority	Level 15 (highest)

[API setting example]

r_cg_main.c

```
/* Start user code for global. Do not edit comment generated here */
extern volatile uint16_t data[16];
volatile uint8_t cnt;
/* End user code. Do not edit comment generated here */

void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    while (1U)
    {
        for (cnt = 0; cnt < 16U; cnt++)
        {
            /* Write new data to compare */
            R_DOC_WriteData(data[cnt]);
        }
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_doc_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint16_t data[16];
volatile uint16_t result;
/* End user code. Do not edit comment generated here */

void r_doc_dopcf_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Get result */
    R_DOC_GetResult((uint16_t *)&result);

    /* Configure the operation mode of DOC */
    R_DOC_SetMode(COMPARE_MISMATCH, 0x0000);

    /* Clear DOPCI flag */
    R_DOC_ClearFlag();
    /* End user code. Do not edit comment generated here */
}
```

3.2.35 Low power timer (LPT)

Below is a list of API functions output by the Code Generator for low power timer use.

Table 3.35 API Functions: [Low power timer]

API Function Name	Function
R_LPT_Create	Performs initialization necessary to control the low power timer.
R_LPT_Create_UserInit	Performs user-defined initialization relating to the low power timer.
R_LPT_Start	Starts counting by a low power timer.
R_LPT_Stop	Ends counting by a low power timer.

R_LPT_Create

Performs initialization necessary to control the low power timer.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_LPT_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LPT_Create_UserInit

Performs user-defined initialization relating to the low power timer.

Remark This API function is called as the R_LPT_Create callback routine.

[Syntax]

```
void    R_LPT_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LPT_Start

Starts counting by a low power timer.

[Syntax]

```
void    R_LPT_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LPT_Stop

Ends counting by a low power timer.

[Syntax]

```
void    R_LPT_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

Usage example

Return from software standby mode to normal operation mode by LPT compare match via event link controller (ELC).

[GUI settings]

Low Power Timer				Used
	LPT			Used
			Low power timer operation setting	Used
			Timer cycle value	8000ms (Actual value: 8000 Erorr: 0%)
			Register value (LPTPRD)	59999
			Compare value 0	29999

Low Power Consumption				Used
	LPC			Used
			LPC operation setting	Ued
			Initial operating power control mode	High-speed operating mode
			Output of the Address bus and bus control signals	Retain the output state in software standby mode
			Initial sleep mode return clock source	Disabled

Event Link Controller				Used
	ELC			Used
		ELC_InterruptLPT		Used
			LPT dedicated interrupt	Used
			Event signal	LPT compare match
			Operation on event	Issues an event to the CPU to wake up from CPU software (highest)
			ELSR8I Priority	Level 15

[API settings]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Enable all ELC event links */
    R_ELC_Start();

    /* Start LPT module */
    R_LPT_Start();

    /* Enable software standby mode */
    R_LPC_SoftwareStandby();

    /* Stop LPT module */
    R_LPT_Stop();

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

3.2.36 Comparator C (CMPC)

Below is a list of API functions output by the Code Generator for Comparator C use.

Table 3.36 API Functions: [Comparator C]

API Function Name	Function
R_CMPC_Create	Performs initialization necessary to control the Comparator C.
R_CMPC_Create_UserInit	Performs user-defined initialization relating to the Comparator C.
r_cmpc_cmpcn_interrupt	Performs processing in response to the comparator C interrupt.
R_CMPCn_Start	Starts comparison for analog input voltage.
R_CMPCn_Stop	Ends comparison for analog input voltage.

R_CMPC_Create

Performs initialization necessary to control the Comparator C.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_CMPC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CMPC_Create_UserInit

Performs user-defined initialization relating to the Comparator C.

Remark This API function is called as the R_CMPC_Create callback routine.

[Syntax]

```
void    R_CMPC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_cmpc_cmpc*n*_interrupt

Performs processing in response to the comparator C interrupt.

Remark This API function is called to run interrupt processing for the comparator C*n* interrupt, which is generated when the comparison result changes at this time.

[Syntax]

```
static void    r_cmpc_cmpcn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CMPC*n*_Start

Starts comparison for analog input voltage.

[Syntax]

```
void    R_CMPCn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CMPC*n*_Stop

Ends comparison for analog input voltage.

[Syntax]

```
void    R_CMPCn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

Usage example

Set the flag to notify changing the comparison result.

[GUI setting example]

Comparator C				Used
	CMPC			Used
			Comparator C0	Used
			Comparator C1	Unused
			Comparator C2	Unused
			Comparator C3	Unused
			Comparator C0 Comparator input select	AN000
			Comparator C0 Reference voltage setting	CVREFC0 (P20)
			Comparator C0 Enable digital filter	Unused
			Comparator C0 Output polarity	Normal
			Comparator C0 Enable output (COMP0)	Unused
			Comparator C0 Enable comparator C0 interrupt (CMPC0)	Used
			Comparator C0 Edge select	Rising
			Comparator C0 Priority	Level 15 (highest)

[API setting example]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start comparator C0 */
    R_CMPC0_Start();

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_cmpc_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_cmpc0_f;
/* End user code. Do not edit comment generated here */

void R_CMPC_Create_UserInit(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Clear the flag */
    g_cmpc0_f = 0U;
    /* End user code. Do not edit comment generated here */
}

static void r_cmpc_cmpc0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Set the flag */
    g_cmpc0_f = 1U;
    /* End user code. Do not edit comment generated here */
}
```

3.2.37 LCD controller / driver (LCD)

Below is a list of API functions output by the Code Generator for LCD controller / driver use.

Table 3.37 API Functions: [LCD controller / driver]

API Function Name	Function
R_LCD_Create	Performs initialization necessary to control the LCD controller / driver.
R_LCD_Create_UserInit	Performs user-defined initialization relating to the LCD controller / driver.
R_LCD_Start	Sets the LCD controller / driver to display on status.
R_LCD_Stop	Sets the LCD controller / driver to display off status.
R_LCD_Voltage_On	Enables operation of internal voltage boost circuit and capacitor split circuit.
R_LCD_Voltage_Off	Disables operation of internal voltage boost circuit and capacitor split circuit.

R_LCD_Create

Performs initialization necessary to control the LCD controller / driver.

Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```
void    R_LCD_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LCD_Create_UserInit

Performs user-defined initialization relating to the LCD controller / driver.

Remark This API function is called as the R_LCD_Create callback routine.

[Syntax]

```
void    R_LCD_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LCD_Start

Sets the LCD controller / driver to display on status.

[Syntax]

```
void    R_LCD_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LCD_Stop

Sets the LCD controller / driver to display off status.

[Syntax]

```
void    R_LCD_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LCD_Voltage_On

Enables operation of internal voltage boost circuit and capacitor split circuit.

[Syntax]

```
void    R_LCD_Voltage_On ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LCD_Voltage_Off

Disables operation of internal voltage boost circuit and capacitor split circuit.

[Syntax]

```
void    R_LCD_Voltage_Off ( void );
```

[Argument(s)]

None.

[Return value]

None.

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	July 01, 2018	—	First Edition issued

Code Generator Tool User's Manual: RL78 API Reference

Publication Date: Rev.1.00 July 01, 2018

Published by: Renesas Electronics Corporation

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

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

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

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

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

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

Renesas Electronics (China) Co., Ltd.
Room 1709 Quantum Plaza, No.27 ZhichunLu, Haidian District, Beijing, 100191 P. R. China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

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

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

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

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

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700, Fax: +91-80-67208777

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

Colophon 5.0

Code Generator Tool

ルネサス エレクトロニクス株式会社

R01UH0001JJ0100