User's Manual

Code Generator

User's Manual: RL78 API Reference

Target Device
RL78 Family

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website (http://www.renesas.com).

Renesas Electronics
www.renesas.com

Rev.1.05 Jul 2024

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,

Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Contact information

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

www.renesas.com/contact/.

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

2. Processing at power-on

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

3. Input of signal during power-off state

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

4. Handling of unused pins

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

5. Clock signals

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

6. Voltage application waveform at input pin

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

7. Prohibition of access to reserved addresses

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

8. Differences between products

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

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 is a software tool that automatically generates device drivers.
This manual gives Output files and API functions.

1.1 Overview

Code Generator 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.
Code Generator not only support Windows operation system, from e^2 studio 2024-07, Linux and Mac OS version also support Code Generator.

1.2 Feature

Code Generatorl 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
  You can output configured information using the Pin Configurator/Code Generator as files in various formats for use as design documents.

- Renaming function [#1]
  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 */

  #1. Only Windows version Code Generator supported renaming function. For Linux and Mac OS version, this function is not support.

1.3 Compiler

The code generated by the Code Generator can be built with the following compilers.

For Windows version
- Renesas compiler (CC-RL, CA78K0R)
- GNU compiler
- IAR compiler
- LLVM compiler

For Linux version #2
- Renesas compiler (CC-RL)
- LLVM compiler

For Mac OS version #2
- LLVM compiler

#2. The Linux and Mac OS version was supported from e^2 studio 2024-07.

1.4 Cautions

Code Generator has the following cautions.

- OSS（Open Source Software）
  The code generation tool does not use OSS.

- Multiple interrupts
  For multiple interrupts, refer to your compilation manual.

- Global variable
  Even if a global variable is initialized by the Create() function of each peripheral, it is cleared by RAM initialization at startup, so the global variable is already cleared when the main() function is executed. (When generating code for Renesas compilers and IAR compilers)

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 (1/15)

Peripheral Function	File Name	API Function Name	output (*1)
Common	r_main.c or r_cg_main.c	main R_MAIN_UserserInit	A A
	r_systeminit.c or r_cg_sysmteminit.c	hdwinit R_Systeminit __low_level_init (*3)	A A A
	r_cg_macrodriver.h	-	-
	r_cg_userdefine.h	-	-
	r_reset_program.asm or r_cg_reset_program.asm (*2) or start.S (*4)	-	-
	r_hardware_setup.c or r_cg_hardware_setup.c (*2, *4)	R_Systeminit HardwareSetup	A A
	r_cg_vector_table.c (*2, *4)	-	-
	r_cg_interrupt_handlers.h (*2, *4)	-	-
	r_cg_config.h	-	-
	r_cg_inthandler.c (*4)	-	-
Clock generator	r_cg_cgc.c	R_CGC_Create R_CGC_Set_ClockMode R_CGC_RAMECC_Start R_CGC_RAMECC_Stop R_CGC_StackPointer_Start R_CGC_StackPointer_Stop R_CGC_ClockMonitor_Start R_CGC_ClockMonitor_Stop	A M A A A A A A
	r_cg_cgc_user.c	R_CGC_Create_UserInit r_cgc_ram_ecc_interrupt r_cgc_stackpointer_interrupt r_cgc_clockmonitor_interrupt R_CGC_Get_ResetSource	M A A A A
	r_cg_cgc.h	-	-
Port functions	r_cg_port.c	R_PORT_Create	A
	r_cg_port_user.c	R_PORT_Create_UserInit	M
	r_cg_port.h	-	-
High-speed on-chip Oscillator clock Frequency Correction function	r_cg_hofc.c	R_HOFC_Create R_HOFC_Start R_HOFC_Stop	A A A
	r_cg_hofc_user.c	R_HOFC_Create_UserInit r_hofc_interrupt	M A
	r_cg_hofc.h	-	-

Table 2.2 Output File List (2/15)

Peripheral Function	File Name	API Function Name	output (*1)
Timer array unit	r_cg_timer.c or r_cg_tau.c	R_TAUm_Create R_TAUm_Channeln_Start R_TAUm_Channeln_Higher8bits_Start R_TAUm_Channeln_Lower8bits_Start R_TAUm_Channeln_Stop R_TAUm_Channeln_Higher8bits_Stop R_TAUm_Channeln_Lower8bits_Stop R_TAUm_Reset R_TAUm_Set_PowerOff R_TAUm_Channeln_Get_PulseWidth R_TAUm_Channeln_Set_SoftwareTriggerOn	A A A A A A A M A A A
	r_cg_timer_user.c or r_cg_tau_user.c	R_TAUm_Create_UserInit r_taum_channeln_interrupt r_taum_channeln_higher8bits_interrupt	M A A
	r_cg_timer.h or r_cg_tau.h	-	-
Timer RJ	r_cg_timer.c or r_cg_tmrj.c	R_TMR_RJn_Create R_TMR_RJn_Start R_TMR_RJn_Stop R_TMR_RJn_Set_PowerOff R_TMR_RJn_Get_PulseWidth R_TMRJn_Create R_TMRJn_Start R_TMRJn_Stop R_TMRJn_Set_PowerOff R_TMRJn_Get_PulseWidth	A A A A A A A A M M
	r_cg_timer_user.c or r_cg_tmrj_user.c	R_TMR_RJn_Create_UserInit r_tmr_rjn_interrupt R_TMRJn_Create_UserInit r_tmrjn_interrupt	M A M A
	r_cg_timer.h or r_cg_tmrj.h	-	-

Table 2.3 Output File List (3/15)

Peripheral Function	File Name	API Function Name	output (*1)
Timer RD	r_cg_timer.c or r_cg_tmrd.c	R_TMR_RDn_Create R_TMR_RDn_Start R_TMR_RDn_Stop R_TMR_RDn_Set_PowerOff R_TMR_RDn_ForcedOutput_Start R_TMR_RDn_ForcedOutput_Stop R_TMR_RDn_Get_PulseWidth R_TMRDn_Create R_TMRDn_Start R_TMRDn_Stop R_TMRDn_Set_PowerOff R_TMRDn_ForcedOutput_Start R_TMRDn_ForcedOutput_Stop R_TMRDn_Get_PulseWidth R_TMRD_Set_PowerOff R_TMRD_PWMOP_ForcedOutput_Stop R_TMRD_PWMOP_Set_PowerOff	A A A M M M A M M M A A A A M A A
	r_cg_timer_user.c or r_cg_tmrd_user.c	R_TMR_RDn_Create_UserInit r_tmr_rdn_interrupt R_TMRDn_Create_UserInit r_tmrdn_interrupt	M A M A
	r_cg_timer.h or r_cg_tmrd.h	-	-
Timer RG	r_cg_timer.c or r_cg_tmrg.c	R_TMR_RG0_Create R_TMR_RG0_Start R_TMR_RG0_Stop R_TMR_RG0_Set_PowerOff R_TMR_RG0_Get_PulseWidth	A A A M A
	r_cg_timer_user.c or r_cg_tmrg_user.c	R_TMR_RG0_Create_UserInit r_tmr_rg0_interrupt	M A
	r_cg_timer.h or r_cg_tmrg.h	-	-
Timer RX	r_cg_tmrx.c	R_TMRX_Create R_TMRX_Start R_TMRX_Stop R_TMRX_Set_PowerOff R_TMRX_Get_BufferValue	A A A A M
	r_cg_tmrx_user.c	R_TMRX_Create_UserInit r_tmrx_interrupt	M A
	r_cg_tmrx.h	-	-

Table 2.4 Output File List (4/15)

Peripheral Function	File Name	API Function Name	output (*1)
16-bit timer KB	r_cg_timer.c or r_cg_tmkb.c	R_TMR_KB_Create	A
		R_TMR_KBm_Start	A
		R_TMR_KBm_Stop	A
		R_TMR_KBm_Set_PowerOff	M
		R_TMR_KBmn_ForcedOutput_Start	A
		R_TMR_KBmn_ForcedOutput_Stop	A
		R_TMR_KBm_BatchOverwriteRequestOn	A
		R_TMR_KBm_ForcedOutput_mn_Start	A
		R_TMR_KBm_ForcedOutput_mn_Stop	A
		R_TMR_KBm_Reset	M
	r_cg_timer_user.c or r_cg_tmkb_user.c	R_TMR_KB_Create_UserInit	M
		r_tmr_kbm_interrupt	A
	r_cg_timer.h or r_cg_tmkb.h	-	-
16-bit timer KC0	r_cg_timer.c	R_TMR_KC0_Create	A
		R_TMR_KC0_Start	A
		R_TMR_KC0_Stop	A
		R_TMR_KC0_Set_PowerOff	M
	r_cg_timer_user.c	R_TMR_KC0_Create_UserInit	M
		r_tmr_kc0_interrupt	A
	r_cg_timer.h	-	-
16-bit timer KB2	r_cg_kb2.c	R_KB2m_Create	A
		R_KB2m_Start	A
		R_KB2m_Stop	A
		R_KB2m_Set_PowerOff	M
		R_KB2m_Simultaneous_Start	A
		R_KB2m_Simultaneous_Stop	A
		R_KB2m_Synchronous_Start	A
		R_KB2m_Synchronous_Stop	A
		R_KB2m_TKBOn0_Forced_Output_Stop_Function1_Start	A
		R_KB2m_TKBOn0_Forced_Output_Stop_Function1_Stop	A
		R_KB2m_TKBOn1_Forced_Output_Stop_Function1_Start	A
		R_KB2m_TKBOn1_Forced_Output_Stop_Function1_Stop	A
		R_KB2m_TKBOn0_DitheringFunction_Start	A
		R_KB2m_TKBOn0_DitheringFunction_Stop	A
		R_KB2m_TKBOn1_DitheringFunction_Start	A
		R_KB2m_TKBOn1_DitheringFunction_Stop	A
		R_KB2m_TKBOn0_SmoothStartFunction_Start	A
		R_KB2m_TKBOn0_SmoothStartFunction_Stop	A
		R_KB2m_TKBOn1_SmoothStartFunction_Start	A
		R_KB2m_TKBOn1_SmoothStartFunction_Stop	A
		R_KB2m_BatchOverwriteRequestOn	A
	r_cg_kb2_user.c	R_KB2m_Create_UserInit	M
		r_kb2m_interrupt	A
	r_cg_kb2.h	-	-

Table 2.5 Output File List (5/15)

Peripheral Function	File Name	API Function Name	output (*1)
Real-time clock	r_cg_rtc.c	R_RTC_Create R_RTC_Start R_RTC_Stop R_RTC_Set_PowerOff R_RTC_Set_HourSystem R_RTC_Set_CounterValue R_RTC_Set_CalendarCounterValue R_RTC_Set_BinaryCounterValue R_RTC_Get_CounterValue R_RTC_Get_CalendarCounterValue R_RTC_Get_BinaryCounterValue R_RTC_Set_ConstPeriodInterruptOn R_RTC_Set_ConstPeriodInterruptOff R_RTC_Set_AlarmOn R_RTC_Set_CalendarAlarmOn R_RTC_Set_BinaryAlarmOn R_RTC_Set_AlarmOff R_RTC_Set_AlarmValue R_RTC_Set_CalenderAlarmValue R_RTC_Set_BinaryAlarmValue R_RTC_Get_AlarmValue R_RTC_Get_CalenderAlarmValue R_RTC_Get_BinaryAlarmValue R_RTC_Set_RTC1HZOn R_RTC_Set_RTC1HZOff R_RTC_Set_RTCOUTOn R_RTC_Set_RTCOUTOff	A A A A M A A A A A A A A A A A A A A A A A A A A A A
	r_cg_rtc_user.c	R_RTC_Create_UserInit r_rtc_interrupt r_rtc_callback_constperiod r_rtc_callback_alarm r_rtc_alarminterrupt r_rtc_periodinterrupt r_rtc_callback_periodic	M A A A A A A
	r_cg_rtc.h	-	-
Subsystem clock frequency measure-ment circuit	r_cg_fmc.c	R_FMC_Create R_FMC_Start R_FMC_Stop R_FMC_Set_PowerOff	A A A M
	r_cg_fmc_user.c	R_FMC_Create_UserInit r_fmc_interrupt	M A
	r_cg_fmc.h	-	-
12-bit interval timer	r_cg_it.c	R_IT_Create R_IT_Start R_IT_Stop R_IT_Reset R_IT_Set_PowerOff	A A A M M
	r_cg_it_user.c	R_IT_Create_UserInit r_it_interrupt	M A
	r_cg_it.h	-	-

Table 2.6 Output File List (6/15)

Peripheral Function	File Name	API Function Name	output (*1)
8-bit interval timer	r_cg_it8bit.c	R_IT8Bitm_Channeln_Create R_IT8Bitm_Channeln_Start R_IT8Bitm_Channeln_Stop R_IT8Bitm_Channeln_Set_PowerOff R_IT8Bitm_Set_PowerOff	A A A M M
	r_cg_it8bit_user.c	R_IT8Bitm_Channeln_Create_UserInit r_it8bitm_channeln_interrupt	M A
	r_cg_it8bit.h	-	-
16-bit wakeup timer	r_cg_timer.c	R_WUTM_Create R_WUTM_Start R_WUTM_Stop R_WUTM_Set_PowerOff	A A A M
	r_cg_timer_user.c	R_WUTM_Create_UserInit r_wutm_interrupt	M A
	r_cg_timer.h	-	-
Clock output/buzzer output controller	r_cg_pclbuz.c	R_PCLBUZn_Create R_PCLBUZn_Start R_PCLBUZn_Stop R_PCLBUZn_Set_PowerOff	A A A M
	r_cg_pclbuz_user.c	R_PCLBUZn_Create_UserInit	M
	r_cg_gpt.h	-	-
Watchdog timer	r_cg_wdt.c	R_WDT_Create R_WDT_Restart	A A
	r_cg_wdt_user.c	R_WDT_Create_UserInit r_wdt_interrupt	M A
	r_cg_wdt.h	-	-
24-bit DS A/D converter with programmable gain instrumentation amplifier	r_cg_pga_dsad.c	R_PGA_DSAD_Create R_PGA_DSAD_Start R_PGA_DSAD_Stop R_PGA_DSAD_Set_PowerOff R_PGA_DSAD_Get_AverageResult R_PGA_DSAD_Get_Result R_PGA_DSAD_CAMP_OffsetTrimming	A A A M A A A
	r_cg_pga_dsad_user.c	R_PGA_DSAD_Create_UserInit r_pga_dsad_interrupt_conversion r_pga_dsad_interrupt_scan r_pga_dsad_conversion_interrupt r_pga_dsad_scan_interrupt	M A A A A
	r_cg_pga_dsad.h	-	-

Table 2.7 Output File List (7/15)

Peripheral Function	File Name	API Function Name	output (*1)
A/D converter	r_cg_adc.c	R_ADC_Create	A
		R_ADC_Set_OperationOn	A
		R_ADC_Set_OperationOff	A
		R_ADC_Start	A
		R_ADC_Stop	A
		R_ADC_Reset	M
		R_ADC_Set_PowerOff	M
		R_ADC_Set_ADChannel	M
		R_ADC_Set_SnoozeOn	M
		R_ADC_Set_SnoozeOff	M
		R_ADC_Set_TestChannel	M
		R_ADC_Get_Result	A
		R_ADC_Get_Result_8bit	A
	r_cg_adc_user.c	R_ADC_Create_UserInit	M
		r_adc_interrupt	A
	r_cg_adc.h	-	-
Configurable amplifier	r_cg_camp.c	R_CAMP_Create	A
		R_CAMPn_Start	A
		R_CAMPn_Stop	A
		R_CAMP_Set_PowerOff	M
	r_cg_camp_user.c	R_CAMP_Create_UserInit	M
	r_cg_camp.h	-	-
Temperature sensor	r_cg_tmps.c	R_TMPS_Create	A
		R_TMPS_Start	A
		R_TMPS_Stop	A
		R_TMPS_Reset	M
		R_TMPS_Set_PowerOff	M
	r_cg_tmps_user.c	R_TMPS_Create_UserInit	M
	r_cg_tmps.h	-	-
24-bit DS A/D converter	r_cg_dsadci.c	R_DSADC_Create	A
		R_DSADC_Set_OperationOn	A
		R_DSADC_Set_OperationOff	A
		R_DSADC_Start	A
		R_DSADC_Stop	A
		R_DSADC_Reset	M
		R_DSADC_Set_PowerOff	M
		R_DSADC_Channeln_Get_Result	A
		R_DSADC_Channeln_Get_Result_16bit	A
	r_cg_dsadc_user.c	R_DSADC_Create_UserInit	M
		r_dsadc_interrupt	A
		r_dsadzcn_interrupt	A
	r_cg_dsadc.h	-	-

Table 2.8 Output File List (8/15)

Peripheral Function	File Name	API Function Name	output (*1)
D/A converter	r_cg_dac.c	R_DAC_Create R_DACn_Start R_DACn_Stop R_DAC_Set_PowerOff R_DACn_Set_ConversionValue R_DACn_Change_OutputVoltage_8bit R_DACn_Change_OutputVoltage R_DACn_Create R_DAC_Reset	A A A M A A A A M
	r_cg_dac_user.c	R_DAC_Create_UserInit R_DACn_Create_UserInit	M M
	r_cg_dac.h	-	-
Programmable gain amplifier	r_cg_pga.c	R_PGA_Create R_PGA_Start R_PGA_Stop R_PGA_Reset R_PGA_Set_PowerOff	A A A M M
	r_pga_user.c	R_PGA_Create_UserInit	M
	r_pga.h	-	-
Comparator	r_cg_comp.c	R_COMP_Create R_COMPn_Start R_COMPn_Stop R_COMP_Reset R_COMP_Set_PowerOff	A A A M M
	r_cg_comp_user.c	R_COMP_Create_UserInit r_compn_interrupt	M A
	r_cg_comp.h	-	-
Comparator/Pro-grammableGain-Amplifier	r_cg_comppgacmpb.c	R_COMPPGA_Create R_COMPPGA_Set_PowerOff R_COMPn_Start R_COMPn_Stop R_PGA_Start R_PGA_Stop R_PWMOPT_Start R_PWMOPT_Stop	A M A A A A A A
	r_cg_comppga_user.c	R_COMPPGA_Create_UserInit r_compn_interrupt	M A
	r_cg_comppga.h	-	-

Table 2.9 Output File List (9/15)

Peripheral Function	File Name	API Function Name	output (*1)
Serial array unit	r_cg_serial.c or r_cg_sau.c	R_SAUm_Create	A
		R_SAUm_Reset	M
		R_SAUm_Set_PowerOff	M
		R_SAUm_Set_SnoozeOn	M
		R_SAUm_Set_SnoozeOff	M
		R_UARTn_Create	A
		R_UARTn_Start	A
		R_UARTn_Stop	A
		R_UARTn_Send	A
		R_UARTn_Receive	A
		R_CSImn_Create	A
		R_CSImn_Start	A
		R_CSImn_Stop	A
		R_CSImn_Send	A
		R_CSImn_Receive	A
		R_CSImn_Send_Receive	A
		R_IICmn_Create	A
		R_IICmn_StartCondition	A
		R_IICmn_StopCondition	A
		R_IICmn_Stop	A
		R_IICmn_Master_Send	A
		R_IICmn_Master_Receive	A
	r_cg_serial_user.c or r_cg_sau_user.c	R_SAUm_Create_UserInit	M
		r_uartn_interrupt_send	A
		r_uartn_interrupt_receive	A
		r_uartn_interrupt_error	A
		r_uartn_callback_sendend	A
		r_uartn_callback_receiveend	A
		r_uartn_callback_error	A
		r_uartn_callback_softwareoverrun	A
		r_csimn_interrupt	A
		r_csimn_callback_sendend	A
		r_csimn_callback_receiveend	A
		r_csimn_callback_error	A
		r_iicmn_interrupt	A
		r_iicmn_callback_master_sendend	A
		r_iicmn_callback_master_receiveend	A
		r_iicmn_callback_master_error	A
	r_cg_serial.h or r_cg_sau.h	-	-
Serial array unit 4	r_cg_serial.c	R_DALIn_Create	A
		R_DALIn_Start	A
		R_DALIn_Stop	A
		R_DALIn_Send	A
		R_DALIn_Receive	A
	r_cg_serial_user.c	r_dalin_interrupt_send	A
		r_dalin_interrupt_receive	A
		r_dalin_interrupt_error	A
		r_dalin_callback_sendend	A
		r_dalin_callback_receiveend	A
		r_dalin_callback_error	A
		r_dalin_callback_softwareoverrun	A
	r_cg_serial.h	-	-

Table 2.10 Output File List (10/15)

Peripheral Function	File Name	API Function Name	output (*1)
Asynchronous serial interface LIN-UART	r_cg_serial.c	R_UARTFn_Create R_UARTFn_Start R_UARTFn_Stop R_UARTFn_Set_PowerOff R_UARTFn_Send R_UARTFn_Receive R_UARTFn_Set_DataComparisonOn R_UARTFn_Set_DataComparisonOff	A A A M A A A A
	r_cg_serial_user.c	R_UARTFn_Create_UserInit r_uartfn_interrupt_send r_uartfn_interrupt_receive r_uartfn_interrupt_error r_uartfn_callback_sendend r_uartfn_callback_receiveend r_uartfn_callback_error r_uartfn_callback_softwareoverrun r_uartfn_callback_expbitdetect r_uartfn_callback_idmatch	M A A A A A A A A A
	r_cg_serial.h	-	-
Serial interface IICA	r_cg_serial.c or r_cg_iica.c	R_IICAn_Create R_IICAn_StopCondition R_IICAn_Stop R_IICAn_Reset R_IICAn_Set_PowerOff R_IICAn_Master_Send R_IICAn_Master_Receive R_IICAn_Slave_Send R_IICAn_Slave_Receive R_IICAn_Set_SnoozeOn R_IICAn_Set_SnoozeOff R_IICAn_Set_WakeupOn R_IICAn_Set_WakeupOff	A M A M M A A A A A A A A
	r_cg_serila_user.c or r_cg_iica_user.c	R_IICAn_Create_UserInit r_iican_interrupt r_iican_callback_master_sendend r_iican_callback_master_receiveend r_iican_callback_master_error r_iican_callback_slave_sendend r_iican_callback_slave_receiveend r_iican_callback_slave_error r_iican_callback_getstopcondition	M A A A A A A A M
	r_cg_serial.h or r_cg_iica.h	-	-
LCD controller/ driver	r_cg_lcd.c	R_LCD_Create R_LCD_Start R_LCD_Stop R_LCD_Set_VoltageOn R_LCD_Set_VoltageOff R_LCD_Set_PowerOff R_LCD_VoltageOn R_LCD_VoltageOff	A A A A A A A A
	r_cg_lcd_user.c	R_LCD_Create_UserInit r_lcd_interrupt	M A
	r_cg_lcd.h	-	-

Table 2.11 Output File List (11/15)

Peripheral Function	File Name	API Function Name	output (*1)
Sound generator	r_cg_sg.c	R_SG_Create R_SG_Start R_SG_Stop	A A A
	r_cg_sg_user.c	R_SG_Create_UserInit r_sg_interrupt	M A
	r_cg_sg.h	-	-
DMA controller	r_cg_dmac.c	R_DMACn_Create R_DMAC_Create R_DMACn_Start R_DMACn_Stop R_DMACn_Set_SoftwareTriggerOn	A A A A A
	r_cg_dmac_user.c	R_DMACn_Create_UserInit R_DMAC_Create_UserInit r_dmacn_interrupt	M M A
	r_cg_dmac.h	-	-
Data transfer controller	r_cg_dtc.c	R_DTC_Create R_DTCn_Start R_DTCn_Stop R_DTC_Set_PowerOff R_DTCDn_Start R_DTCDn_Stop	A A A M A A
	r_cg_dtc_user.c	R_DTC_Create_UserInit	M
	r_cg_dtc.h	-	-
Event link controller	r_cg_elc.c	R_ELC_Create R_ELC_Stop	A A
	r_cg_elc_user.c	R_ELC_Create_UserInit	M
	r_cg_elc.h	-	-
Interrupt functions	r_cg_intc.c	R_INTC_Create R_INTCn_Start R_INTCn_Stop R_INTCLRn_Start R_INTCLRn_Stop R_INTRTCICn_Start R_INTRTCICn_Stop R_INTFO_Start R_INTFO_Stop R_INTFO_ClearFlag	A A A A A A A A A A
	r_cg_intc_user.c	R_INTC_Create_UserInit r_intcn_interrupt r_intclrn_interrupt r_intrtcicn_interrupt r_intfo_interrupt	M A A A A
	r_cg_intc.h	-	-

Table 2.12 Output File List (12/15)

Peripheral Function	File Name	API Function Name	output (*1)
Key interrupt function	r_cg_intc.c or r_cg_key.c	R_KEY_Create R_KEY_Start R_KEY_Stop	A A A
	r_cg_key_user.c	R_KEY_Create_UserInit r_key_interrupt	M A
	r_cg_opamp.h	-	-
Voltage detector	r_cg_doc.c	R_LVD_Create R_LVD_InterruptMode_Start R_LVD_Start_VDD R_LVD_Start_VBAT R_LVD_Start_VRTC R_LVD_Start_EXLVD R_LVD_Stop_VDD R_LVD_Stop_VBAT R_LVD_Stop_VRTC R_LVD_Stop_EXLVD R_LVI_Create R_LVI_InterruptMode_Start	A A A A A A A A A A A A
	r_cg_doc_user.c	R_LVD_Create_UserInit r_lvd_interrupt r_lvd_vddinterrupt r_lvd_vbatinterrupt r_lvd_vrtcinterrupt r_lvd_exlvdinterrupt R_LVI_Create_UserInit r_lvi_interrupt	M A A A A A M A
	r_cg_doc.h	-	-
Battery backup function	r_cg_bup.c	R_BUP_Create R_BUP_Start R_BUP_Stop	A A A
	r_cg_bupt_user.c	R_BUP_Create_UserInit r_bup_interrupt	M A
	r_cg_bupt.h	-	-
Oscillation stop detector	r_cg_osdc.c	R_OSDC_Create R_OSDC_Start R_OSDC_Stop R_OSDC_Set_PowerOff R_OSDC_Reset	A A A M M
	r_cg_osdc_user.c	R_OSDC_Create_UserInit r_osdc_interrupt	M A
	r_cg_osdc.h	-	-
SPI interface	r_cg_saic.c or r_cg_spi.c	R_SAIC_Create R_SAIC_Write R_SAIC_Read R_SPI_Create R_SPI_Write R_SPI_Read	A A M A A A
	r_cg_saic_user.c or r_cg_spi_user.c	R_SAIC_Create_UserInit R_SPI_Create_UserInit	M M
	r_cg_saic.h or r_cg_spi.h	-	-

Table 2.13 Output File List (13/15)

Peripheral Function	File Name	API Function Name	output (*1)
Operational amplifier	r_cg_opamp.c	R_OPAMP_Create	A
		R_OPAMP_Set_ReferenceCircuitOn	A
		R_OPAMP_Set_ReferenceCircuitOff	A
		R_OPAMPn_Start	A
		R_OPAMPn_Stop	A
		R_OPAMPn_Set_PrechargeOn	M
		R_OPAMPn_Set_PrechargeOff	M
	r_cg_opamp_user.c	R_OPAMP_Create_UserInit	M
	r_cg_opamp.h	-	-
Data operation circuit	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_DOC_Set_PowerOff	M
		R_DOC_Reset	M
	r_cg_doc_user.c	R_DOC_Create_UserInit	M
		r_doc_interrupt	A
	r_cg_doc.h	-	-
32-bit Multiply-accumulator	r_cg_mac32bit.c	R_MAC32Bit_Create	A
		R_MAC32Bit_Reset	M
		R_MAC32Bit_Set_PowerOff	M
		R_MAC32Bit_MULUnsigned	M
		R_MAC32Bit_MULSigned	M
		R_MAC32Bit_MACUnsigned	M
		R_MAC32Bit_MACSigned	M
	r_cg_mac32bit_user.c	R_MAC32Bit_Create_UserInit	M
		r_mac32bit_interrupt_flow	A
	r_cg_mac32bit.h	-	-
12-bit A/D converter	r_cg_12adc.c	R_12ADC_Create	A
		R_12ADC_Start	A
		R_12ADC_Stop	A
		R_12ADC_Get_ValueResult	A
		R_12ADC_Set_ADChannel	M
		R_12ADC_TemperatureSensorOutput_On	M
		R_12ADC_TemperatureSensorOutput_Off	M
		R_12ADC_InternalReferenceVoltage_On	M
		R_12ADC_InternalReferenceVoltage_Off	M
		R_12ADC_Set_PowerOff	M
	r_cg_12adc_user.c	R_12ADC_Create_UserInit	M
		r_12adc_interrupt	A
	r_cg_12adc.h	-	-
12-bit D/A converter	r_cg_12da.c	R_12DA_Create	A
		R_12DAn_Start	A
		R_12DAn_Stop	A
		R_12DA_Set_PowerOff	M
		R_12DAn_Set_ConversionValue	A
	r_cg_12adc_user.c	R_12DA_Create_UserInit	M
	r_cg_12adc.h	-	-

Table 2.14 Output File List (14/15)

Peripheral Function	File Name	API Function Name	output (*1)
Operational amplifier and Analog switch	r_cg_ampansw.c	R_AMPANSW_Create R_OPAMPm_Set_ReferenceCircuitOn R_OPAMPm_Set_ReferenceCircuitOff R_OPAMPm_Start R_OPAMPm_Stop R_ANSW_ChargePumpm_On R_ANSW_ChargePumpm_Off	A A A A A A A
	r_cg_ampansw_user.c	R_AMPANSW_Create_UserInit	M
	r_cg_ampansw.h	-	-
Voltage reference	r_cg_vr.c	R_VR_Create R_VR_Start R_VR_Stop	A A A
	r_cg_vr_user.c	R_VR_Create_UserInit	M
	r_cg_vr.h	-	-
Sampling output timer detector	r_cg_smotd.c	R_SMOTD_Create R_SMOTD_Start R_SMOTD_Stop R_SMOTD_Set_PowerOff	A A A M
	r_cg_smotd_user.c	R_SMOTD_Create_UserInit r_smotd_counterA_interrupt r_smotd_counterB_interrupt r_smotd_smpn_interrupt	M A A A
	r_cg_smotd.h	-	-
External signal sampler	r_cg_exsd.c	R_EXSD_Create R_EXSD_Start R_EXSD_Stop R_EXSD_Set_PowerOff	A A A M
	r_cg_exsd_user.c	R_EXSD_Create_UserInit r_exsd_interrupt	M A
	r_cg_exsd.h	-	-
Serial interface UARTMG	r_cg_uartmg.c	R_UARTMGn_Create R_UARTMGn_Start R_UARTMGn_Stop R_UARTMGn_Set_PowerOff R_UARTMGn_Send R_UARTMGn_Receive	A A A M A A
	r_cg_uartmg_user.c	R_UARTMGn_Create_UserInit r_uartmgn_interrupt_send r_uartmgn_interrupt_receive r_uartmgn_interrupt_error r_uartmgn_callback_sendend r_uartmgn_callback_receiveend r_uartmgn_callback_error r_uartmgn_callback_softwareoverrun	M A A A A A A A
	r_cg_uartmg.h	-	-

Table 2.15 Output File List (15/15)

Peripheral Function	File Name	API Function Name	output (*1)
Amplifier unit	r_cg_amp.c	R_AMP_Create R_AMP_Set_PowerOn R_AMP_Set_PowerOff R_PGA1_Start R_PGA1_Stop R_AMPn_Start R_AMPn_Stop	A A A A A A A
	r_cg_amp_user.c	R_AMP_Create_UserInit	M
	r_cg_ampr.h	-	-
Data flash libraries	r_cg_fdl.c	R_FDL_Create R_FDL_Open R_FDL_Close R_FDL_Write R_FDL_Read R_FDL_Erase	A A A A A A
	r_cg_fdl.h	-	-

*1 In case of [API output control] setting is 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.

*2 When generating code for the GNU compiler

*3 When generating code for the IAR compiler

*4 When generating code for the LLVM compiler

3. API FUNCITONS

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.

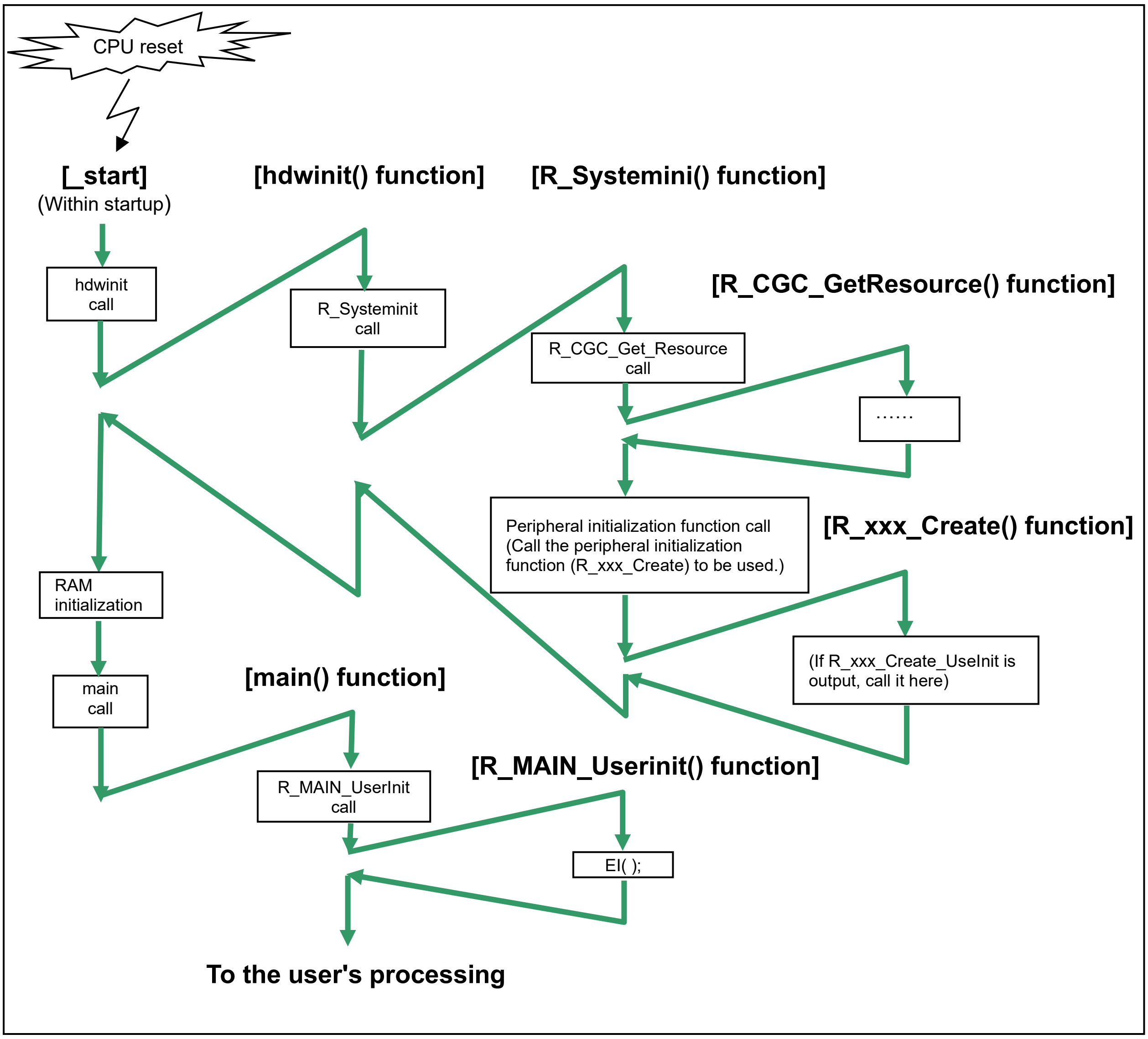
Note: The code generated by the code registrar includes functions that use the for statement, while statement, and do while statement (loop processing) to wait for register settings. If fail-safe processing for an infinite loop is required, check the generated code and add processing.

3.2 Initialization process

This section describes the initialization flow up to the main() function.

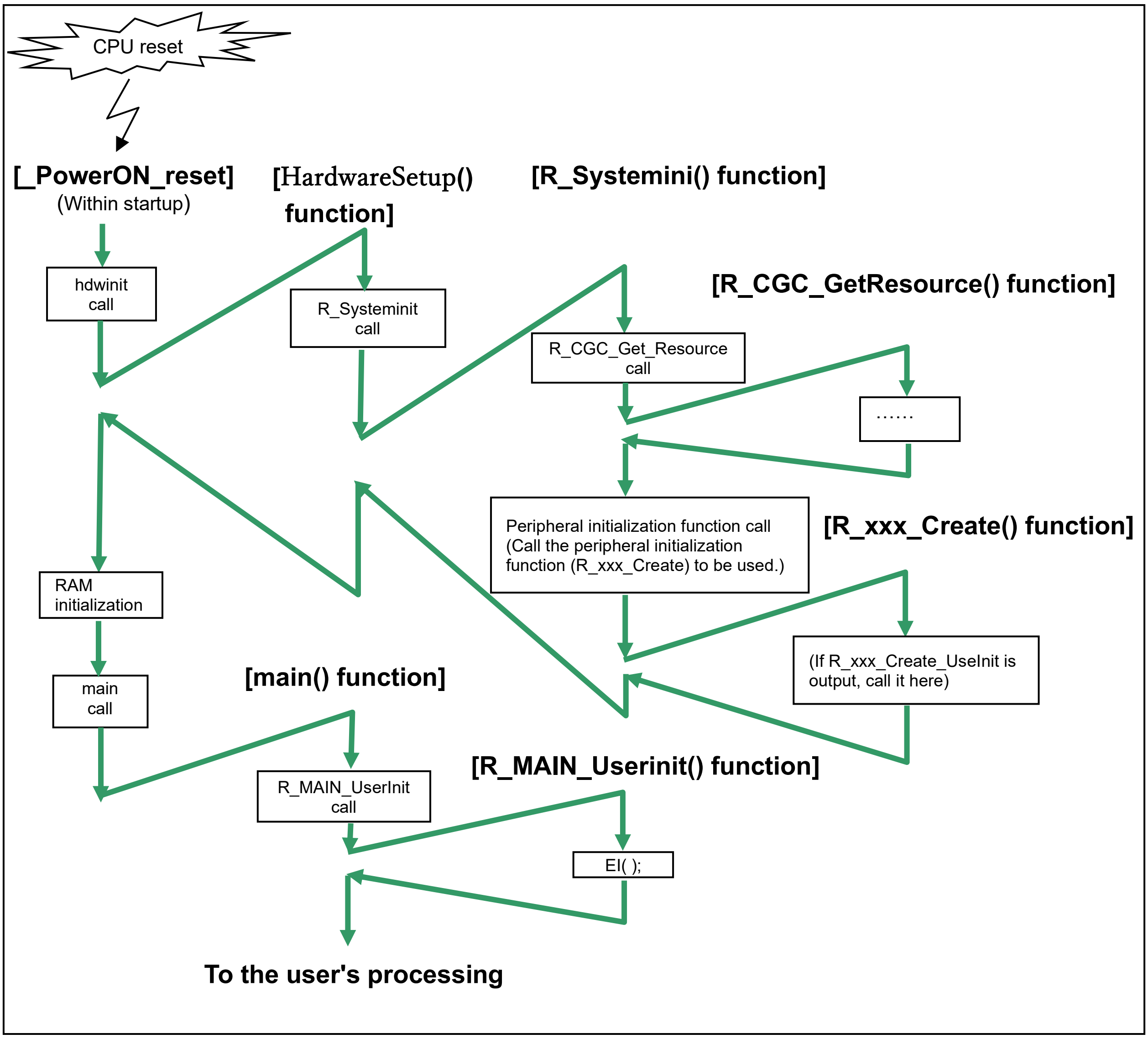
3.2.1 For Renesas compiler

Figure 3.1 Initial flow for Renesas compiler



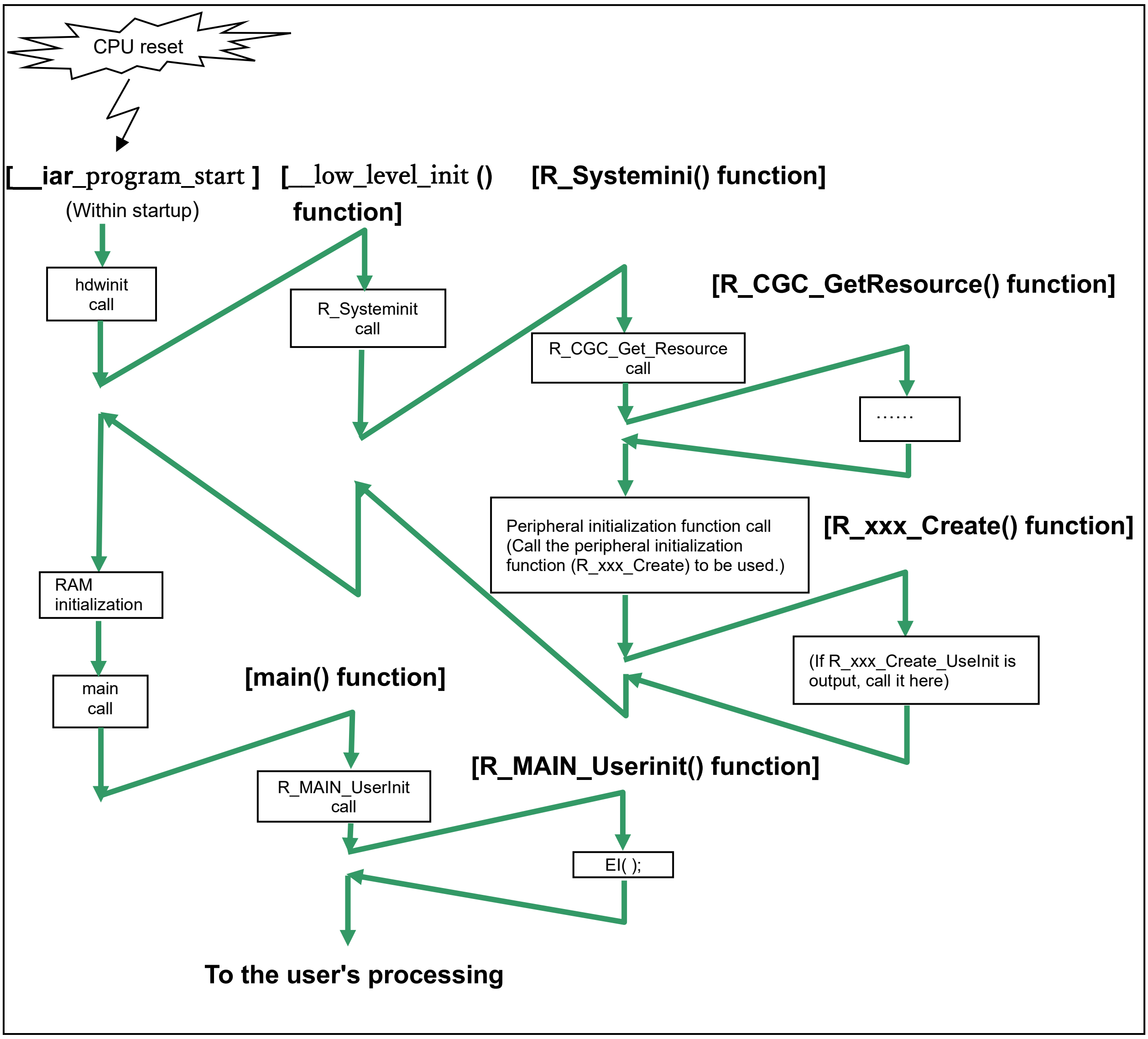
3.2.2 For GNU/LLVM compiler

Figure 3.2 Initial flow for GNU/LLVM compiler



3.2.3 For IAR compiler

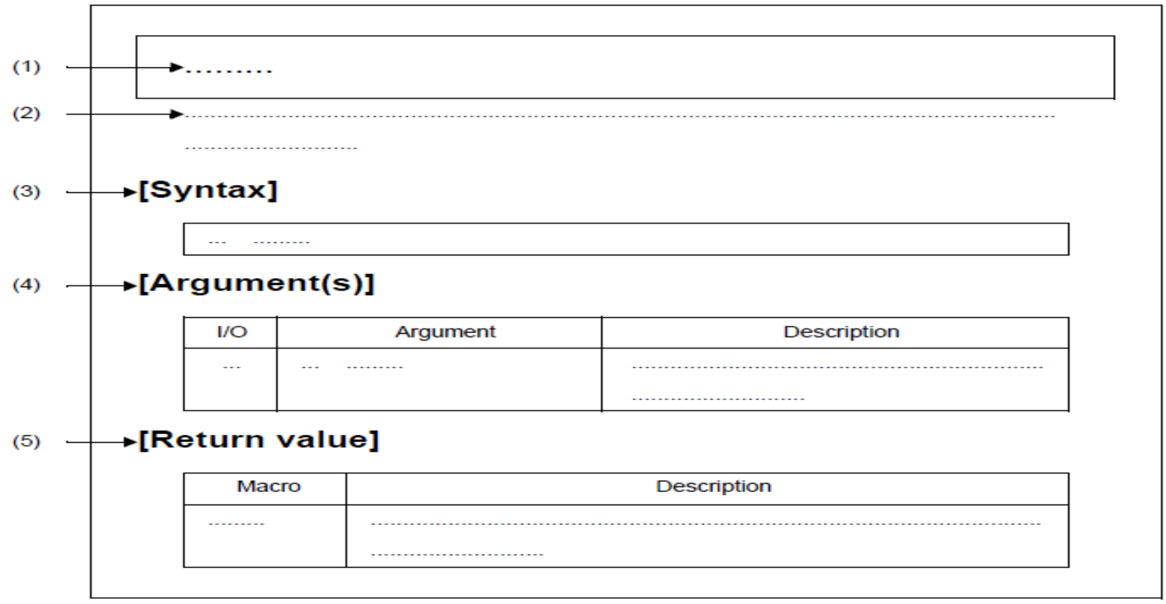
Figure 3.3 Initial flow for IAR compiler



3.3 Function Reference

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

Figure 3.4 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	Description
(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) [Retuen 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

3.3.1 Common

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

Table 3.1 API Functions: [Common]

API Function Name	Function
hdwinit	Performs initialization necessary to control the various hardwares. This API is automatically called from the startup routine of Renesas compiler.
_low_level_init	Performs initialization necessary to control the various hardwares. This API is automatically called from the startup routine of IAR compiler.
HardwareSetup	Performs initialization necessary to control the various hardwares. This API is automatically called from the startup routine (r_reset_program.asm) of Renesas compiler.
R_Systeminit	Performs initialization necessary to control the various peripheral functions.
main	This is a main function.
R_MAIN_UserserInit	Performs user-defined initialization.

hdwinit

__low_level_init

HardwareSetup

Performs initialization necessary to control the various hardwares.

Remark This API function is called from the startup routine.

[Syntax]

```
void        hdwinit ( void );
```

```
int         __low_level_init( void );
```

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

[Argument(s)]

None.

[Return value]

hdwint is none.

Value	Description
1U	Normal completion

R_Systeminit

Performs initialization necessary to control the various peripheral functions.

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

main

This is a main function.

Remark Call this API function from the startup 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.

3.3.2 Clock generator

Below is a list of API functions output by the Code Generator for clock generator (include reset function, on-chip debug function, etc.) use.

Table 3.2 API Functions: [Clock Generator]

API Function Name	Function
R_CGC_Create	Performs initialization required to control the clock generator (include reset function, on-chip debug function, etc.).
R_CGC_Create_UserInit	Performs user-defined initialization relating to the clock generator (include reset function, on-chip debug function, etc.).
r_cgc_ram_ecc_interrupt	Performs processing in response to the RAM 1-bit correction/2-bit error detection interrupt INTRAM.
r_cgc_stackpointer_interrupt	Performs processing in response to the stackpointer overflow/underflow interrupt INTSPM.
r_cgc_clockmonitor_interrupt	Performs processing in response to the clock monitor interrupt INTCLM.
R_CGC_Get_ResetSource	Performs processing in response to RESET signal.
R_CGC_Set_ClockMode	Changes the CPU clock/peripheral hardware clock.
R_CGC_RAMECC_Start	Starts the RAM-ECC function.
R_CGC_RAMECC_Stop	Ends the RAM-ECC function.
R_CGC_StackPointer_Start	Starts the CPU stack pointer monitor function.
R_CGC_StackPointer_Stop	Ends the CPU stack pointer monitor function.
R_CGC_ClockMonitor_Start	Starts the clock monitor.
R_CGC_ClockMonitor_Stop	Ends the clock monitor.

R_CGC_Create

Performs initialization required to control the clock generator (include reset function, on-chip debug function, etc.).

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 generator (include reset function, on-chip debug function, etc.).

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_ram_ecc_interrupt

Performs processing in response to the RAM 1-bit correction/2-bit error detection interrupt INTRAM.

Remark This API function is called as the interrupt process corresponding to the RAM 1-bit correction/2-bit error detection interrupt INTRAM.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_cgc_ram_ecc_interrupt ( void );
```

CC-RL Compiler

```
static void     __near r_cgc_ram_ecc_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_cgc_stackpointer_interrupt

Performs processing in response to the stack pointer overflow/underflow interrupt INTSPM.

Remark This API function is called as the interrupt process corresponding to the stack pointer overflow/underflow interrupt INTSPM.

[Syntax]

CA78K0R Compiler

```
__interrupt static void r_cgc_stackpointer_interrupt ( void );
```

CC-RL Compiler

```
static void __near r_cgc_stackpointer_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_cgc_clockmonitor_interrupt

Performs processing in response to the clock monitor interrupt INTCLM.

Remark This API function is called as the interrupt process corresponding to the clock monitor interrupt INTCLM.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_cgc_clockmonitor_interrupt ( void );
```

CC-RL Compiler

```
static void    __near r_cgc_clockmonitor_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CGC_Get_ResetSource

Performs processing in response to RESET signal.

[Syntax]

```
void        R_CGC_Get_ResetSource ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CGC_Set_ClockMode

Changes the CPU clock/peripheral hardware clock.

[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 generator type HIOCLK : High-speed onchip oscillator SYSX1CLK : X1 clock SYSEXTCLK : External main system clock SUBXT1CLK : XT1 clock SUBEXTCLK : External subsystem clock

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Exit with error (abend)
MD_ERROR2	Exit with error (abend)
MD_ERROR3	Exit with error (abend)
MD_ERROR4	Exit with error (abend)
MD_ARGERROR	Invalid argument specification

R_CGC_RAMECC_Start

Starts the RAM-ECC function.

[Syntax]

```
void        R_CGC_RAMECC_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CGC_RAMECC_Stop

Ends the RAM-ECC function.

[Syntax]

```
void        R_CGC_RAMECC_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CGC_StackPointer_Start

Starts the CPU stack pointer function.

[Syntax]

```
void        R_CGC_StackPointer_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CGC_StackPointer_Stop

Ends the CPU stack pointer function.

[Syntax]

```
void        R_CGC_StackPointer_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CGC_ClockMonitor_Start

Starts the clock monitor.

[Syntax]

```
void        R_CGC_ClockMonitor_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CGC_ClockMonitor_Stop

Ends the clock monitor.

[Syntax]

```
void        R_CGC_ClockMonitor_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

Usage example

Switch clock by external input.

[GUI setting example]

Clock Generator				Used
	CGC			Used
			Operation mode setting	High speed min mode 4.0(V) ≦ VDD ≦ 5.5(V)
			Main system clock (fMAIN) setting	High-speed OCO(fIH)
			fIH Operation	Used
			fIH Frequency	64(MHz)
			fMX Operation	Used
			High-speed system clock setting	X1 oscillation(fX)
			fMX frequency	4(MHz)
			Stable time	65536 (2^18/fX) (μs)
			fPLL operation	Unused
			Main/PLL select clock (fMP) setting	64 (fMAIN)(MHz)
			fSUB operation	Used
			Subsystem clock (fSUB) setting	XT1oscillation(fXT)
			fSUB frequency	32.768(kHz)
			XT1 oscillator oscillation mode setting	Low power consumption
			STOP	Subsystem clock in STOP, HALT mode setting
			Internal low-speed oscillation clock (fIL) setting	15(kHz)
			Low speed on-chip oscillatior clock (fSL) setting	32.768 (fSUB)(kHz)
			WDT operation clock (fWDT) setting	15(kHz)
			RTCoperation clock setting	32.768 (fSUB)(kHz)
			Timer RD operation clock	64000 (fIH)(kHz)
			CPU and peripheralclock setting (fCLK)	32000 (fMP/2) (kHz)

Interrupt				Used
	INTP			Used
		INTP0		
			Valid edge	Falling
			Priority	Low

[API setting example]

r_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Clear INTP0 interrupt flag and enable interrupt */
    R_INTC0_Start();

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

r_cg_intc_user.c

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

/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_cgc_f = 0U;
/* End user code. Do not edit comment generated here */

static void __near r_intc0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Change clock generator operation mode */
    if (0U == g_cgc_f)
    {
        if (MD_OK == R_CGC_Set_ClockMode(SUBXT1CLK))
        {
            g_cgc_f = 1U;
        }
    }
    else
    {
        if (MD_OK == R_CGC_Set_ClockMode(HIOCLK))
        {
            g_cgc_f = 0U;
        }
    }
    /* End user code. Do not edit comment generated here */
}
```

3.3.3 Port functions

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

Table 3.3 API Functions: [Port Functions]

API Function Name	Function
R_PORT_Create	Performs initialization necessary to control the port functions.
R_PORT_Create_UserInit	Performs user-defined initialization relating to the port functions.

R_PORT_Create

Performs initialization necessary to control the port functions.

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 port functions.

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.

3.3.4 High-speed on-chip Oscillator clock Frequency Correction function

Below is a list of API functions output by the Code Generator for the High-speed on-chip Oscillator clock Frequency Correction function use.

Table 3.4 API Functions: [High-speed on-chip Oscillator clock Frequency Correction function]

API Function Name	Function
R_HOFC_Create	Performs initialization necessary to control the High-speed on-chip Oscillator clock Frequency Correction function.
R_HOFC_Create_UserInit	Performs user-defined initialization relating to the High-speed on-chip Oscillator clock Frequency Correction function.
r_hofc_interrupt	Performs processing in response to the timer interrupt.
R_HOFC_Start	Starts the count for the High-speed on-chip Oscillator clock Frequency Correction function.
R_HOFC_Stop	Ends the count for the High-speed on-chip Oscillator clock Frequency Correction function.

R_HOFC_Create

Performs initialization necessary to control the High-speed on-chip Oscillator clock Frequency Correction function.

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

[Syntax]

```
void        R_HOFC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_HOFC_Create_UserInit

Performs user-defined initialization relating to the High-speed on-chip Oscillator clock Frequency Correction function.

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

[Syntax]

```
void        R_HOFC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_hofc_interrupt

Performs processing in response to the timer interrupt.

Remark This API function is called as the interrupt process corresponding to the High-speed on-chip Oscillator clock Frequency Correction function interrupt INTCR.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_hofc_interrupt ( void );
```

CC-RL Compiler

```
static void     _near r_hofc_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_HOFC_Start

Starts the count for the High-speed on-chip Oscillator clock Frequency Correction function.

[Syntax]

```
void        R_HOFC_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_HOFC_Stop

Ends the count for the High-speed on-chip Oscillator clock Frequency Correction function.

[Syntax]

```
void        R_HOFC_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.3.5 Timer array unit

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

Table 3.5 API Functions: [Timer Array Unit]

API Function Name	Function
R_TAUm_Create	Performs initialization necessary to control the timer array unit.
R_TAUm_Create_UserInit	Performs user-defined initialization relating to the timer array unit.
r_taum_channeln_interrupt	Performs processing in response to the timer interrupt INTTM*mn*.
r_taum_channeln_higher8bits_interrupt	Performs processing in response to the timer interrupt INTTM*mn*H.
R_TAUm_Channeln_Start	Starts the count for channel *n*.
R_TAUm_Channeln_Higher8bits_Start	Starts the count (higher 8-bit) for channel *n*.
R_TAUm_Channeln_Lower8bits_Start	Starts the count (lower 8-bit) for channel *n*.
R_TAUm_Channeln_Stop	Ends the count for channel *n*.
R_TAUm_Channeln_Higher8bits_Stop	Ends the count (higher 8-bit) for channel *n*.
R_TAUm_Channeln_Lower8bits_Stop	Ends the count (lower 8-bit) for channel *n*.
R_TAUm_Reset	Reset the timer array unit.
R_TAUm_Set_PowerOff	Halts the clock supplied to the timer array unit.
R_TAUm_Channeln_Get_PulseWidth	Captures the high/low-level width measured between pulses of the signal (pulses) input to the TI*mn* pin.
R_TAUm_Channeln_Set_SoftwareTriggerOn	Generates the trigger (software trigger) for one-shot pulse output.

R_TAU*m*_Create

Performs initialization necessary to control the timer array unit.

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

[Syntax]

```
void        R_TAUm_Create ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Create_UserInit

Performs user-defined initialization relating to the timer array unit.

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

[Syntax]

```
void    R_TAUm_Create_UserInit ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_tau*m*_channel*n*_interrupt

Performs processing in response to the timer interrupt INTTM*mn*.

Remark This API function is called as the interrupt process corresponding to the timer interrupt INTTM*mn*.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_taum_channeln_interrupt ( void );
```

CC-RL Compiler

```
static void    _near r_taum_channeln_interrupt ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_tau*m*_channel*n*_higher8bits_interrupt

Performs processing in response to the timer interrupt INTTM*mn*H.

Remark This API function is called as the interrupt process corresponding to the timer interrupt INTTM*mn*H.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_taum_channeln_higher8bits_interrupt ( void );
```

CC-RL Compiler

```
static void    _near r_taum_channeln_higher8bits_interrupt ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Channel*n*_Start

Starts the count for channel *n*.

Remark 1. The time from the call to this API function to the start of counting depends on the type of the function in question (e.g. interval timer, square-wave output, or external event counter).

Remark 2. If the timer is stopped and restarted (again, R_TAUm_Channeln_Start), the counter value is reloaded from the TDR register to the TCR register. For that reason, the timer is set to the initial value.

[Syntax]

```
void        R_TAUm_Channeln_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Channel*n*_Higher8bits_Start

Starts the count (higher 8-bit) for channel *n*.

Remark This API function can only be called when the timer array unit is used as an 8-bit timer.

[Syntax]

```
void R_TAUm_Channeln_Higher8bits_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Channel*n*_Lower8bits_Start

Starts the count (lower 8-bit) for channel *n*.

Remark 1. This API function can only be called when the timer array unit is used as an 8-bit timer.

Remark 2. The time from the call to this API function to the start of counting depends on the type of the function in question (e.g. interval timer, external event counter, or delay counter).

[Syntax]

```
void    R_TAUm_Channeln_Lower8bits_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Channel*n*_Stop

Ends the count for channel *n*.

Remark If the timer is stopped and restarted (again, R_TAUm_Channeln_Start), the counter value is reloaded from the TDR register to the TCR register. For that reason, the timer is set to the initial value.

[Syntax]

```
void        R_TAUm_Channeln_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Channel*n*_Higher8bits_Stop

Ends the count (higher 8-bit) for channel *n*.

Remark This API function can only be called when the timer array unit is used as a 8-bit timer.

[Syntax]

```
void    R_TAUm_Channeln_Higher8bits_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Channel*n*_Lower8bits_Stop

Ends the count (lower 8-bit) for channel *n*.

Remark This API function can only be called when the timer array unit is used as a 8-bit timer.

[Syntax]

```
void R_TAUm_Channeln_Lower8bits_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Reset

Reset the timer array unit.

[Syntax]

```
void        R_TAUm_Reset ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Set_PowerOff

Halts the clock supplied to the timer array unit.

Remark Calling this API function changes the timer array unit to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_TAUm_Set_PowerOff ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TAU*m*_Channel*n*_Get_PulseWidth

Captures the high/low-level width measured between pulses of the signal (pulses) input to the TI*mn* pin.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void        R_TAUm_Channeln_Get_PulseWidth ( uint32_t * const width );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	Unit32_t * const *width*;	Pointer to an area to store the measurement width (0x0 to 0x1FFFF)

[Return value]

None.

R_TAU*m*_Channel*n*_Set_SoftwareTriggerOn

Generates the trigger (software trigger) for one-shot pulse output.

[Syntax]

```
void    R_TAUm_Channeln_Set_SoftwareTriggerOn ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

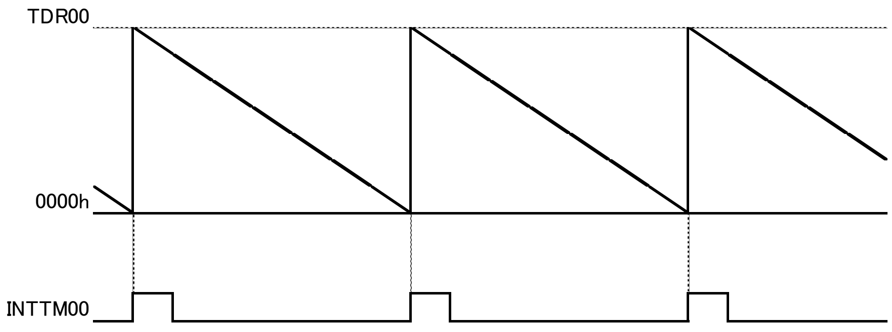
[Return value]

None.

Usage example (Interval timer)

Enter the interrupt function at fixed intervals and count the number of interrupt occurrence.

[Waveform example]



[GUI setting example]

Timer				Used
	TAU0			Used
		Channel0		
			Channel 0	Interval timer
			Intervalvalue (16 bits)	100μs (Actual value：100)
			Generates INTTM00 when counting is started	Unused
			End of timer channel0 count, generate an interrupt (INTTM00)	Used
			Priority (INTTM00)	Low

[API setting example]

r_main.c

```
void main(void)
{
	R_MAIN_UserInit();
	/* Start user code. Do not edit comment generated here */
	/* Start TAU0 channel 0 counter */
	R_TAU0_Channel0_Start();

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

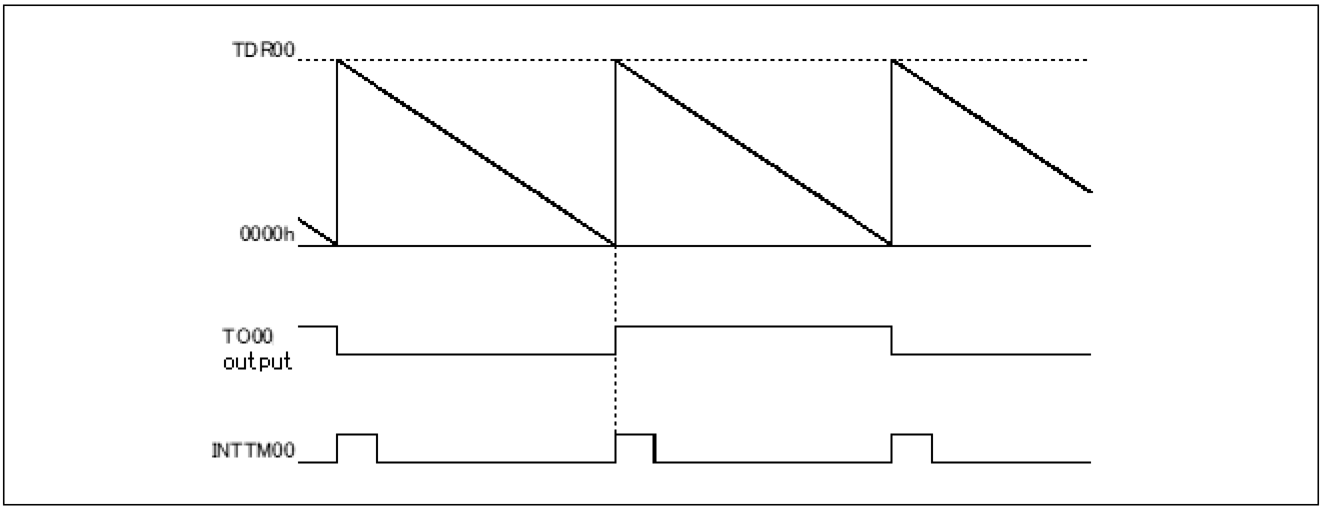
r_cg_timer_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_cnt = 0U;
/* End user code. Do not edit comment generated here */

static void __near r_tau0_channel0_interrupt(void)
{
	/* Start user code. Do not edit comment generated here */
	/* Count INTTM00 */
	g_cnt++;
	/* End user code. Do not edit comment generated here */
}
```

Usage example (Square wave output)

Perform toggle operation at fixed intervals and output a suquare wave with a duty factor of 50%.

[Waveform example]



[GUI setting example]

Timer				Used
	TAU0			Used
		Channel0		
			channel 0	Square wave output
			Square width	100μs (Actual value： 100)
			Generates INTTM00 and inverts timer output when counting is started	Unused
			Initial output value	0
			End of timer channel0 count, generate an interrupt (INTTM00)	Used
			Priority (INTTM00)	Low

[API setting example]

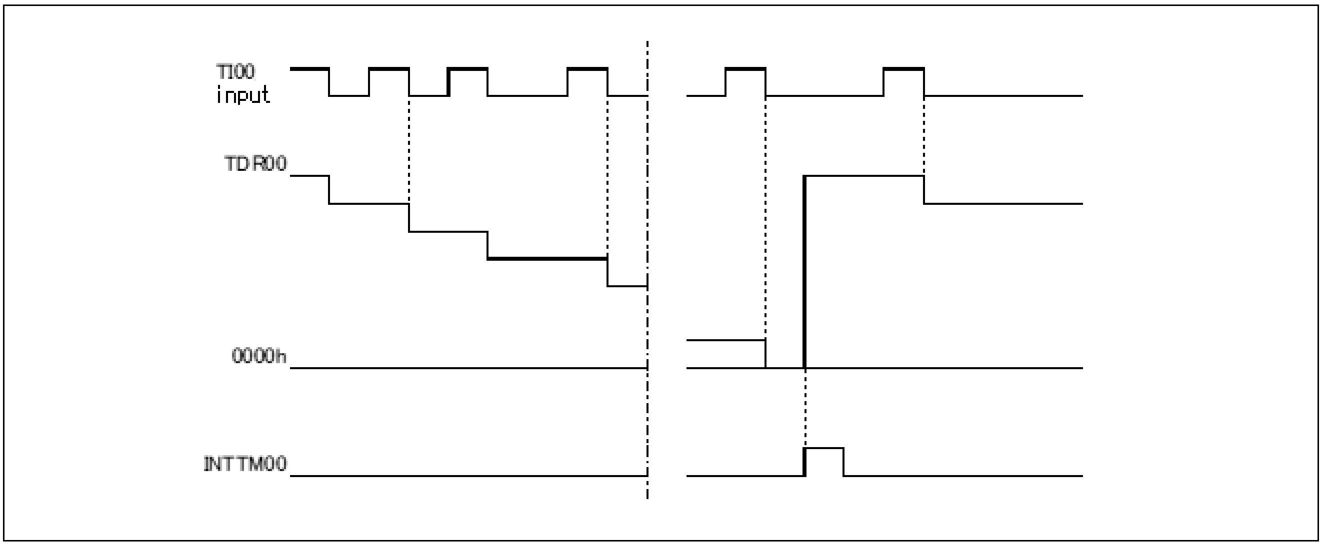
r_main.c

```
void main(void)
{
	R_MAIN_UserInit();
	/* Start user code. Do not edit comment generated here */
	/* Start TAU0 channel 0 counter */
	R_TAU0_Channel0_Start();

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

Usage example (External event counter)

Count up to 100 falling edges.

[Waveform example]



[GUI setting example]

Timer				Used
	TAU0			Used
		Channel0		
			channel 0	External event counter
			TI00 muximum frequency	16000000 (Hz)
			Enable using noise filter of TI00 pin input signal	Unused
			External event select	TI00 falling edge
			Count value	100
			End of timer channel 0 count, generate an interrupt (INTTM00)	Used
			Priority	Low

[API setting example]

r_main.c

```
void main(void)
{
        R_MAIN_UserInit();
        /* Start user code. Do not edit comment generated here */
        /* Start TAU0 channel 0 counter */
        R_TAU0_Channel0_Start();

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

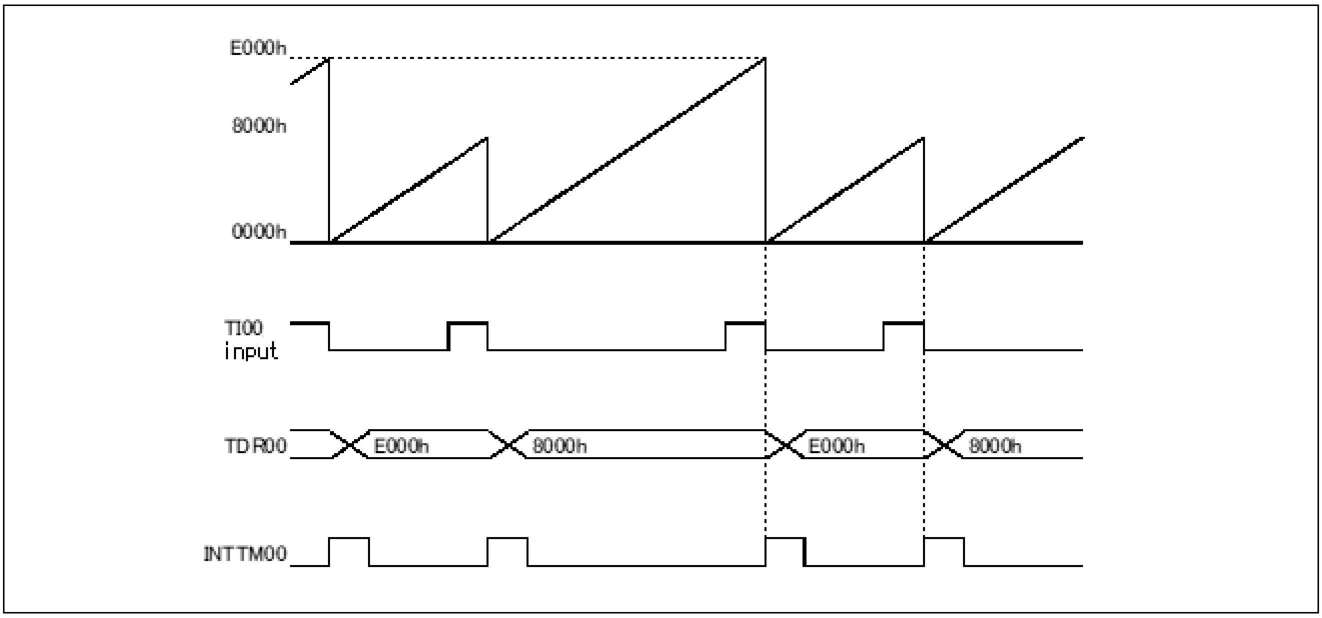
r_cg_timer_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_cnt = 0U;
/* End user code. Do not edit comment generated here */

static void __near r_tau0_channel0_interrupt(void)
{
        /* Start user code. Do not edit comment generated here */
        /* Count INTTM00 */
        g_cnt++;
        /* End user code. Do not edit comment generated here */
}
```

Usage example (Input pulse interval measuremen)

Measure the interval of the falling edges input to TImn pin.

[Waveform example]



[GUI setting example]

Timer				Used
	TAU0			Used
		Channel 0		
			channel 0	Input pulse interval measurement
			Input source setting	TI00
			TI00 interval range	0.125 (μs) < TI00 < 8.192 (ms)
			Enable using noise filter of TI00 pin input signal	Unused
			Generates INTTM00when counting is started.	Unused
			Input edge setting	Falling edge
			End of timer channel0 capture, generate an interrupt (INTTM00)	Used
			Priority	Low

Remark The period of count clock is 1/2 of the minimum value in the eage selected ‘TI00 interval range’. At this GUI setting example, the period of count clock is 0.0625usec.

[API setting example]

r_main.c

```
void main(void)
{
	R_MAIN_UserInit();
	/* Start user code. Do not edit comment generated here */
	/* Start TAU0 channel 0 counter */
	R_TAU0_Channel0_Start();

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

r_cg_timer_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint32_t g_width = 0UL;
/* End user code. Do not edit comment generated here */

static void __near r_tau0_channel0_interrupt(void)
{
	if ((TSR00 & _0001_TAU_OVERFLOW_OCCURS) == 1U)	/* overflow occurs */
	{
		g_tau0_ch0_width = (uint32_t)(TDR00 + 1U) + 0x10000U;
	}
	else
	{
		g_tau0_ch0_width = (uint32_t)(TDR00 + 1U);
	}

	/* Start user code. Do not edit comment generated here */
	/* Get TAU0 channel 0 input pulse width. Pulse width(usec) = (Period of count clock(usec) *
g_width) */
	R_TAU0_Channel0_Get_PulseWidth((uint32_t *)&g_width);
	/* End user code. Do not edit comment generated here */
}
```

3.3.6 Timer RJ

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

Table 3.6 API Functions: [Timer RJ]

API Function Name	Function
R_TMR_RJn_Create	Performs initialization necessary to control the 16-bit timer RJ*n*.
R_TMR_RJn_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer RJ*n*.
r_tmr_rjn_interrupt	Performs processing in response to the timer interrupt.
R_TMR_RJn_Start	Starts the count for 16-bit timer RJ*n*.
R_TMR_RJn_Stop	Ends the count for 16-bit timer RJ*n*.
R_TMR_RJn_Set_PowerOff	Halts the clock supplied to the 16-bit timer RJ*n*.
R_TMR_RJn_Get_PulseWidth	Reads the pulse width of the 16-bit timer RJ*n*.
R_TMRJn_Create	Performs initialization necessary to control the 16-bit timer RJ*n*.
R_TMRJn_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer RJ*n*.
r_tmrjn_interrupt	Performs processing in response to the timer interrupt.
R_TMRJn_Start	Starts the count for 16-bit timer RJ*n*.
R_TMRJn_Stop	Ends the count for 16-bit timer RJ*n*.
R_TMRJn_Set_PowerOff	Halts the clock supplied to the 16-bit timer RJ*n*.
R_TMRJn_Get_PulseWidth	Reads the pulse width of the 16-bit timer RJ*n*.

R_TMR_RJ*n*_Create

Performs initialization necessary to control the 16-bit timer RJ*n*.

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

[Syntax]

```
void R_TMR_RJn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RJ*n*_Create_UserInit

Performs user-defined initialization relating to the 16-bit timer RJ*n*.

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

[Syntax]

```
void R_TMR_RJn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_tmr_rj*n*_interrupt

Performs processing in response to the timer interrupt.

Remark This API function is called as the interrupt process corresponding to the timer interrupt.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_tmr_rjn_interrupt ( void );
```

CC-RL Compiler

```
static void     __near r_tmr_rjn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RJ*n*_Start

Starts the count for 16-bit timer RJ*n*.

[Syntax]

```
void    R_TMR_RJn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RJ*n*_Stop

Ends the count for 16-bit timer RJ*n*.

[Syntax]

```
void    R_TMR_RJn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RJ*n*_Set_PowerOff

Halts the clock supplied to the 16-bit timer RJ*n*.

Remark Calling this API function changes the 16-bit timer RJ*n* to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_TMR_RJn_Set_PowerOff ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RJ*n*_Get_PulseWidth

Reads the pulse width of the 16-bit timer RJ*n*.

Remark 1. This API function can only be called when the 16-bit timer RJ*n* is being used for pulse width measurement mode / pulse period measurement mode.

Remark 2. If there is an overflow (2 pulses or more) during pulse-width measurement, then the pulse width will not be read correctly.

Remark 3. The data obtained at the first interrupt is invalid because it returns the difference from the initial value.

[Syntax]

```
#include   "r_cg_macrodriver.h"
void       R_TMR_RJn_Get_PulseWidth ( uint32_t * const active_width );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	unit32_t * const *active_width*;	Pointer to an area storing the active level width that was read from the TRJ*n*IO pin

[Return value]

None.

R_TMRJ*n*_Create

Performs initialization necessary to control the 16-bit timer RJ*n*.

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

[Syntax]

```
void        R_TMRJn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMRJ*n*_Create_UserInit

Performs user-defined initialization relating to the 16-bit timer RJ*n*.

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

[Syntax]

```
void        R_TMRJn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_tmrj*n*_interrupt

Performs processing in response to the timer interrupt.

Remark This API function is called as the interrupt process corresponding to the timer interrupt.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_tmrjn_interrupt ( void );
```

CC-RL Compiler

```
static void    __near r_tmrjn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMRJ*n*_Start

Starts the count for 16-bit timer RJ*n*.

[Syntax]

```
void    R_TMRJn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMRJ*n*_Stop

Ends the count for 16-bit timer RJ*n*.

[Syntax]

```
void    R_TMRJn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMRJ*n*_Set_PowerOff

Halts the clock supplied to the 16-bit timer RJ*n*.

Remark Calling this API function changes the 16-bit timer RJ*n* to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_TMRJn_Set_PowerOff ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMRJ*n*_Get_PulseWidth

Reads the pulse width of the 16-bit timer RJ*n*.

Remark 1. This API function can only be called when the 16-bit timer RJ*n* is being used for pulse width measurement mode / pulse period measurement mode.

Remark 2. If there is an overflow (2 pulses or more) during pulse-width measurement, then the pulse width will not be read correctly.

Remark 3. The data obtained at the first interrupt is invalid because it returns the difference from the initial value.

[Syntax]

```
#include   "r_cg_macrodriver.h"
void       R_TMRJn_Get_PulseWidth ( uint32_t * const active_width );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	unit32_t * const *active_width*;	Pointer to an area storing the active level width that was read from the TRJ*n*IO pin

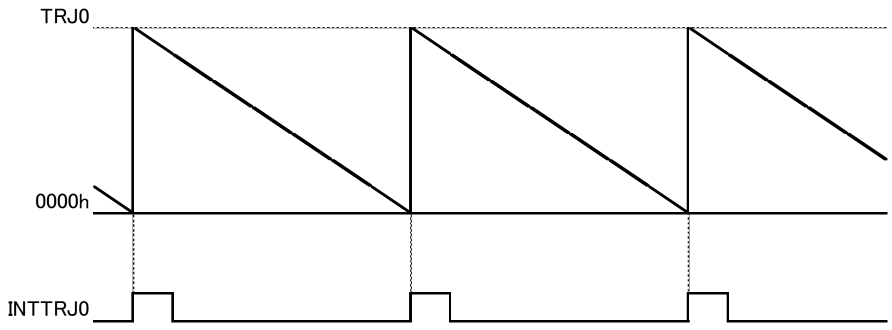
[Return value]

None.

Usage example (Timer mode)

Enter the interrupt function at fixed intervals and count the number of interrupt occurrence.

[Waveform example]



[GUI setting example]

Timer				Used
	TMRJ0			Used
			Functions	Timer mode
			Count source setting	Auto
			Timer value	100µs (Actual value：100)
			When the counter underflows, generate an interrupt (INTTRJ0)	Used
			Priority	Low

[API setting example]

r_main.c

```
void main(void)
{
	R_MAIN_UserInit();
	/* Start user code. Do not edit comment generated here */
	/* Start TMRJ0 counter */
	R_TMR_RJ0_Start();

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

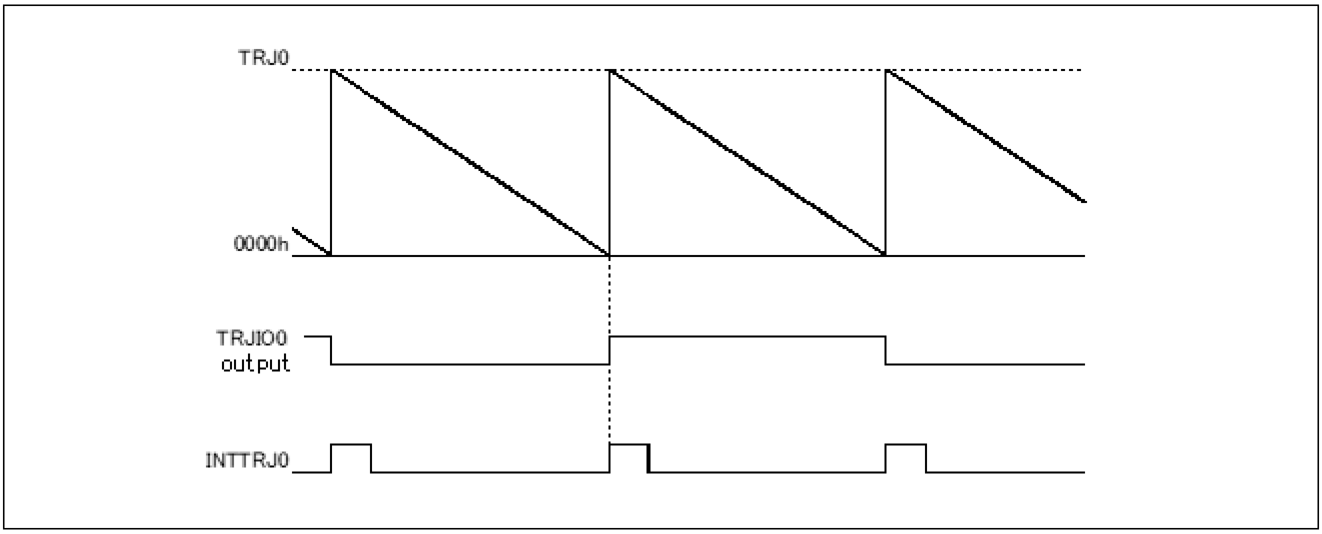
r_cg_timer_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_cnt = 0U;
/* End user code. Do not edit comment generated here */

static void __near r_tmr_rj0_interrupt(void)
{
	/* Start user code. Do not edit comment generated here */
	/* Count INTTRJ0 */
	g_cnt++;
	/* End user code. Do not edit comment generated here */
}
```

Usage example (Pulse output mode)

Perform toggle operation at fixed intervals and output a suquare wave with a duty factor of 50%.

[Waveform example]



[GUI setting example]

Timer				Used
	TMRJ0			Used
			Functions	Pulse output mode
			Count source setting	Auto
			Timer value	100μs (Actual value：100)
			Output (TRIJO0)	Starts as “H”
			Enable output(TRJO0)	Unused
			When the counter underflows, generate an interrupt (INTTRJ0)	Used
			Priority	Low

[API setting example]

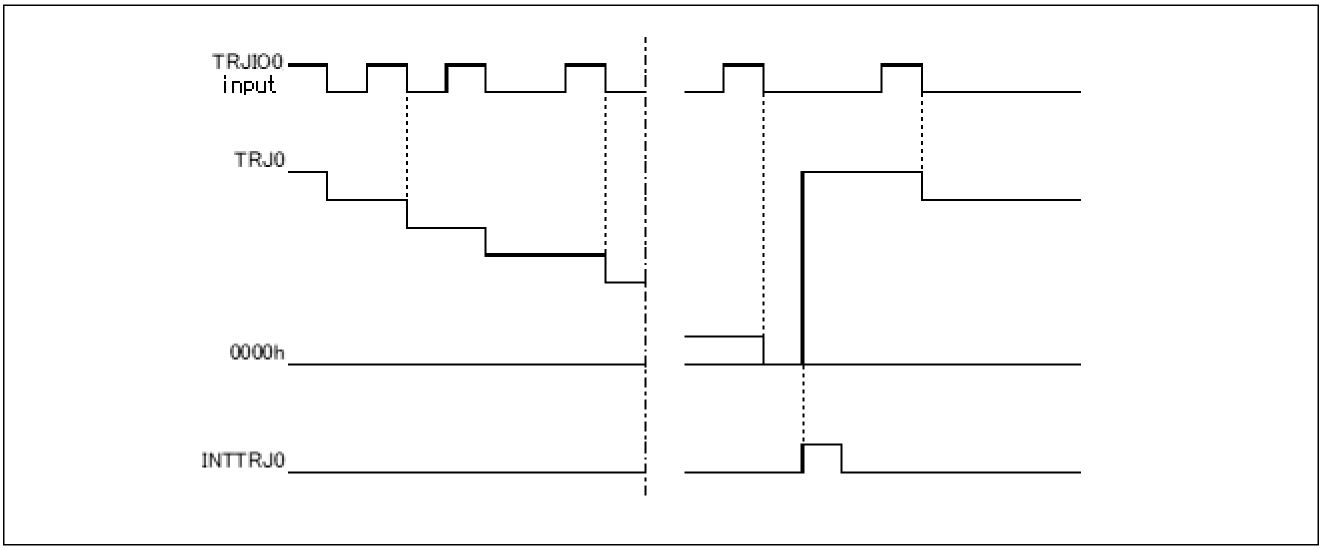
r_main.c

```
void main(void)
{
	R_MAIN_UserInit();
	/* Start user code. Do not edit comment generated here */
	/* Start TMRJ0 counter */
	R_TMR_RJ0_Start();

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

Usage example (Event counter mode)

Count up to 100 falling edges.

[Waveform example]



[GUI setting example]

Timer				Used
	TMRJ0			Used
			Funcitons	Event counter mode
			Count value	100
			TRJIO0 input filter used	Unused
			TRJIO0 event input	Always enable
			TRJIO0 input polarity setting	One edge
			TRJIO polarity switch setting	Starts counting at falling edge of the TRJIO0 input and TRJO0 starts output at “H”.
			Enable output (TRJO0)	Unused
			When the counter underflows, generate an interrupt(INTTRJ0)	Used
			Priority	Low

[API setting example]

r_main.c

```
void main(void)
{
	R_MAIN_UserInit();
	/* Start user code. Do not edit comment generated here */
	/* Start TMRJ0 counter */
	R_TMR_RJ0_Start();

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

r_cg_timer_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_cnt = 0U;
/* End user code. Do not edit comment generated here */

static void __near r_tmr_rj0_interrupt(void)
{
	/* Start user code. Do not edit comment generated here */
	/* Count INTTRJ0 */
	g_cnt++;
	/* End user code. Do not edit comment generated here */
}
```

3.3.7 Timer RD

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

Table 3.7 API Functions: [Timer RD]

API Function Name	Function
R_TMR_RDn_Create	Performs initialization necessary to control the 16-bit timer RD*n*.
R_TMR_RDn_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer RD*n*.
r_tmr_rdn_interrupt	Performs processing in response to the timer interrupt.
R_TMR_RDn_Start	Starts the count for 16-bit timer RD*n*.
R_TMR_RDn_Stop	Ends the count for 16-bit timer RD*n*.
R_TMR_RDn_Set_PowerOff	Halts the clock supplied to the 16-bit timer RD*n*.
R_TMR_RDn_ForcedOutput_Start	Starts the pulse output forced cutoff for 16-bit timer RD*n*,
R_TMR_RDn_ForcedOutput_Stop	Ends the pulse output forced cutoff for 16-bit timer RD*n*.
R_TMR_RDn_Get_PulseWidth	Reads the pulse width of the 16-bit timer RD*n*.
R_TMRDn_Create	Performs initialization necessary to control the 16-bit timer RD*n*.
R_TMRDn_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer RD*n*.
r_tmrdn_interrupt	Performs processing in response to the timer interrupt.
R_TMRDn_Start	Starts the count for 16-bit timer RD*n*.
R_TMRDn_Stop	Ends the count for 16-bit timer RD*n*.
R_TMRDn_Set_PowerOff	Halts the clock supplied to the 16-bit timer RD*n*.
R_TMRDn_ForcedOutput_Start	Starts the pulse output forced cutoff for 16-bit timer RD*n*,
R_TMRDn_ForcedOutput_Stop	Ends the pulse output forced cutoff for 16-bit timer RD*n*.
R_TMRDn_Get_PulseWidth	Reads the pulse width of the 16-bit timer RD*n*.
R_TMRD_Set_PowerOff	Halts the clock supplied to the 16-bit timer RD.
R_TMRD_PWMOP_ForcedOutput_Stop	Ends the PWM output forced cutoff for 16-bit timer RD.
R_TMRD_PWMOP_Set_PowerOff	Halts the clock supplied to the PWM option unit 16-bit timer RD.

R_TMR_RD*n*_Create

Performs initialization necessary to control the 16-bit timer RD*n*.

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

[Syntax]

```
void R_TMR_RDn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RD*n*_Create_UserInit

Performs user-defined initialization relating to the 16-bit timer RD*n*.

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

[Syntax]

```
void        R_TMR_RDn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_tmr_rd*n*_interrupt

Performs processing in response to the timer interrupt.

Remark This API function is called as the interrupt process corresponding to the timer interrupt.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_tmr_rdn_interrupt ( void );
```

CC-RL Compiler

```
static void    __near r_tmr_rdn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RD*n*_Start

Starts the count for 16-bit timer RD*n*.

[Syntax]

```
void    R_TMR_RDn_Start ( void );
```

Remark   *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RD*n*_Stop

Ends the count for 16-bit timer RD*n*.

[Syntax]

```
void    R_TMR_RDn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RD*n*_Set_PowerOff

Halts the clock supplied to the 16-bit timer RD*n*.

Remark Calling this API function changes the 16-bit timer RD*n* to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_TMR_RDn_Set_PowerOff ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RD*n*_ForcedOutput_Start

Starts the pulse output forced cutoff for 16-bit timer RD*n*.

[Syntax]

```
void    R_TMR_RDn_ForcedOurput_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RD*n*_ForcedOutput_Stop

Ends the pulse output forced cutoff for 16-bit timer RD*n*.

Remark This API function can only be called when the 16-bit timer RD*n* is the count to stopped (the TSTART bit in the timer RD start register (TRDSTR) is 0).

[Syntax]

```
void    R_TMR_RDn_ForcedOutput_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_RD*n*_Get_PulseWidth

Reads the pulse width of the 16-bit timer RD*n*.

Remark 1. This API function can only be called when the 16-bit timer RDn is being used for input capture function.

Remark 2. If there is an overflow (2 pulses or more) during pulse-width measurement, then the pulse width will not be read correctly.

[Syntax]

```
#include        "r_cg_macrodriver.h"
#include        "r_cg_timer.h"
MD_STATUS   R_TMR_RDn_Get_PulseWidth ( uint32_t * const active_width,
  uint32_t * const inactive_width, timer_channel_t channel );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	unit32_t * const *active_width*;	Pointer to an area storing the active level width that was read
O	uint32_t * const *inactive_width*;	Pointer to an area storing the inactive level width that was read
I	timer_channel_t *channel*;	Pin to read TMCHANNELA : TRDIOA*n* pin TMCHANNELB : TRDIOB*n* pin TMCHANNELC : TRDIOC*n* pin TMCHANNELD : TRDIOD*n* pin

[Return value]

Macro	Description
MD_OK	Normal completion

R_TMRD*n*_Create

Performs initialization necessary to control the 16-bit timer RD*n*.

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

[Syntax]

```
void        R_TMRDn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMRD*n*_Create_UserInit

Performs user-defined initialization relating to the 16-bit timer RD*n*.

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

[Syntax]

```
void    R_TMRDn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_tmrd*n*_interrupt

Performs processing in response to the timer interrupt.

Remark This API function is called as the interrupt process corresponding to the timer interrupt.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_tmrdn_interrupt ( void );
```

CC-RL Compiler

```
static void    __near r_tmrdn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMRD*n*_Start

Starts the count for 16-bit timer RD*n*.

[Syntax]

```
void    R_TMRDn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMRD*n*_Stop

Ends the count for 16-bit timer RD*n*.

[Syntax]

```
void    R_TMRDn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMRD*n*_Set_PowerOff

Halts the clock supplied to the 16-bit timer RD*n*.

Remark Calling this API function changes the 16-bit timer RD*n* to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_TMRDn_Set_PowerOff ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMRD*n*_ForcedOutput_Start

Starts the pulse output forced cutoff for 16-bit timer RD*n*.

[Syntax]

```
void    R_TMRDn_ForcedOurput_Start ( void );
```

Remark   *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMRD*n*_ForcedOutput_Stop

Ends the pulse output forced cutoff for 16-bit timer RD*n*.

Remark This API function can only be called when the 16-bit timer RD*n* is the count to stopped (the TSTART bit in the timer RD start register (TRDSTR) is 0).

[Syntax]

```
void    R_TMRDn_ForcedOutput_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMRD*n*_Get_PulseWidth

Reads the pulse width of the 16-bit timer RD*n*.

Remark 1. This API function can only be called when the 16-bit timer RD*n* is being used for input capture function.

Remark 2. If there is an overflow (2 pulses or more) during pulse-width measurement, then the pulse width will not be read correctly.

[Syntax]

```
#include        "r_cg_macrodriver.h"
#include        "r_cg_timer.h"
MD_STATUS   R_TMRDn_Get_PulseWidth ( uint32_t * const active_width,
  uint32_t * const inactive_width, timer_channel_t channel );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	unit32_t * const *active_width*;	Pointer to an area storing the active level width that was read
O	uint32_t * const *inactive_width*;	Pointer to an area storing the inactive level width that was read
I	timer_channel_t *channel*;	in to read TMCHANNELA : TRDIOA*n* pin TMCHANNELB : TRDIOB*n* pin TMCHANNELC : TRDIOC*n* pin TMCHANNELD : TRDIOD*n* pin

[Return value]

Macro	Description
MD_OK	Normal completion

R_TMRD_Set_PowerOff

Halts the clock supplied to the 16-bit timer RD.

Remark Calling this API function changes the 16-bit timer RDn to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_TMRD_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMRD_PWMOP_ForcedOutput_Stop

Releases the PWM output forced cutoff for 16-bit timer RD.

[Syntax]

```
void    R_TMRD_PWMOP_ForcedOutput_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMRD_PWMOP_Set_PowerOff

Halts the clock supplied to the PWM option unit 16-bit timer RD.

[Syntax]

```
void    R_TMRD_PWMOP_Set_PowerOff ( void );
```

[Argument(s)]

None.

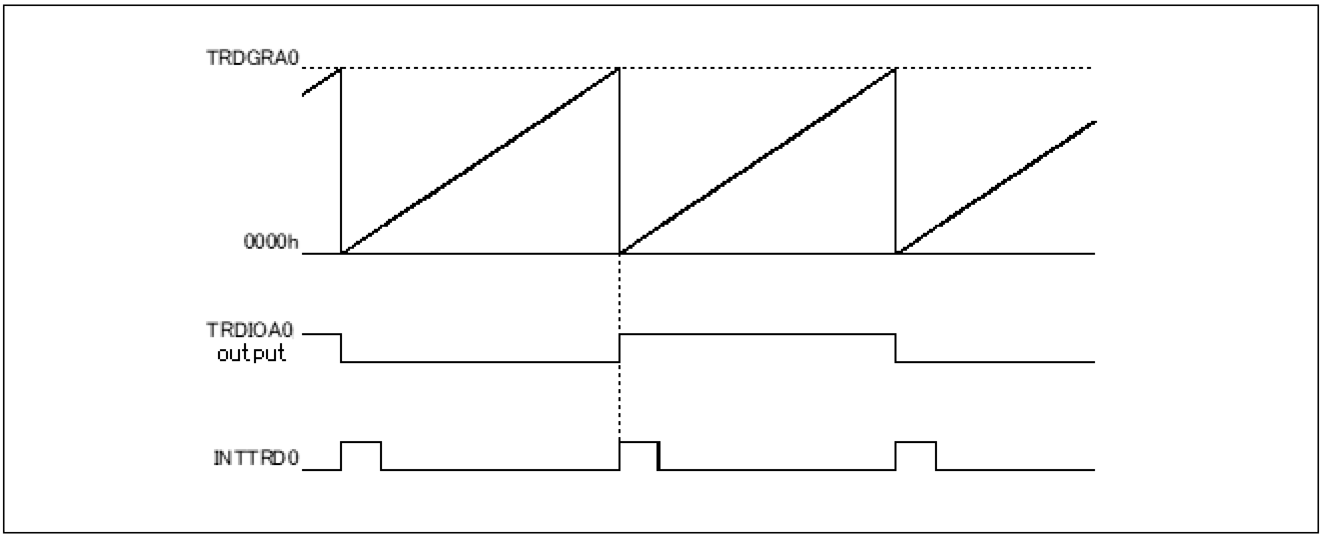
[Return value]

None.

Usage example (Output compare function)

Perform toggle operation at fixed intervals and output a suquare wave with a duty factor of 50%.

[Waveform example]



[GUI setting example]

Timer				Used
	TMRD0			Used
			Output compare function	Used
			Count source setting	Internal clock
			Internal clock setting	fIH
			Counter operation	Count continues at TRDGRA0 comparematch
			Counter clear	Clear by TRDGRA0 compare match
			Regiaster function setting(TRDGRC0)	General register
			Regiaster function setting (TRDGRD0)	General register
			Compare value setting TRDGRA0	100(μs)(Actual value：100)
			Output setting TRDIOA0 pin	Initial output “L” Compare match Toggle output
			Enable TRDGRA0 compare match interrupt	Used

[API setting example]

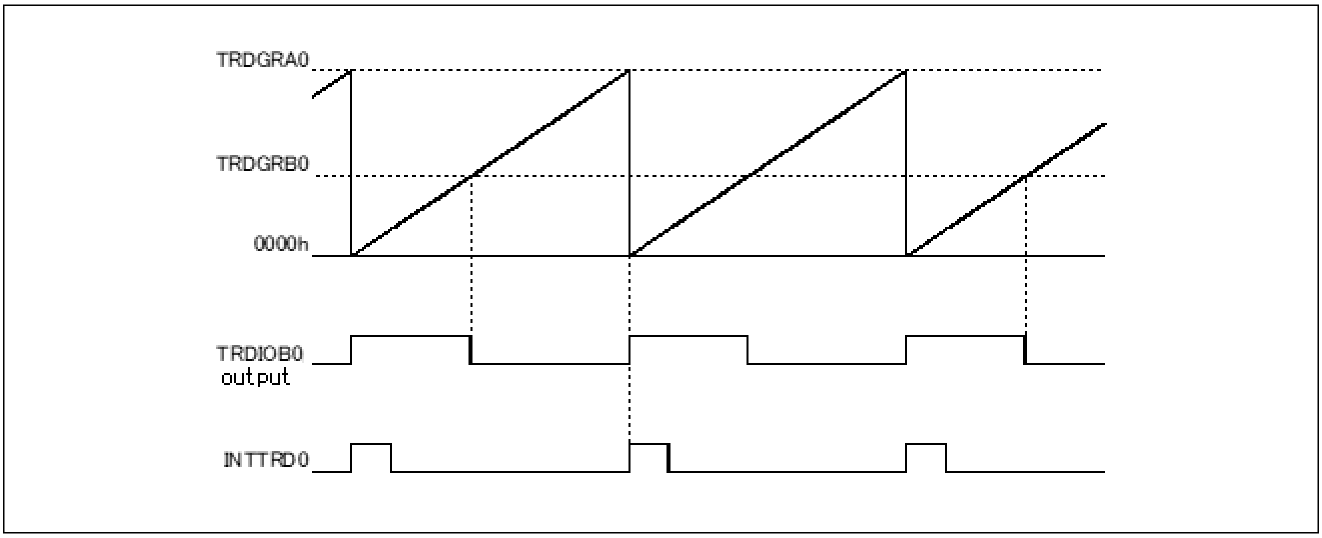
r_main.c

```
void main(void)
{
        R_MAIN_UserInit();
        /* Start user code. Do not edit comment generated here */
        /* Start TMRD0 counter */
        R_TMR_RD0_Start();

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

Usage example (PWM mode (up to 3 PWM outputs))

Output PWM function with specified cycle and duty.

[Waveform example]



[GUI setting example]

Timer				Used
	TMRD0			Used
			PWM mode (up to 3 PWM outputs)	Used
			Count source setting	Internal clock
			Internal clock setting	fIH
			Counter opeartion	Count continues at TRDGRA0 compare match
			Register function setting (TRDGRC0)	General register
			Register function setting (TRDGRD0)	General register
			PWM period	100 (μs) (Actual value：100)
			Duty (TRDGRB0)	50(%)(Actual value：50)
			Output delay time (TRDGRB0)	No delay
			Initial output (TRDIOB0pin)	Non-active level
			Output level (TRDIOB0 pin)	“L” active
			Enable forced cutoff by ELC event input	Unused
			Enable forced cutoff by INTP0 low-level-input	Unused
			TRDIOB0 pin output	Forced cutoff disabled
			TRDIOC0 pin output	Forced cutoff disabled
			TRDIOD0 pin output	Forced cutoff disabled

			Enable TRDGRA0 compare match interrupt	Used
			Enable TRDGRB0 compare match interrupt	Unused
			Enable TRDGRC0 compare match interrupt	Unused
			Enable TRDGRD0 compare match interrupt	Unused
			Enable TRD0 overflow intrrupt	Unused
			Priority	Low

[API setting example]

r_main.c

```
void main(void)
{
        R_MAIN_UserInit();
        /* Start user code. Do not edit comment generated here */
        /* Start TMRD0 counter */
        R_TMR_RD0_Start();

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

3.3.8 Timer RG

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

Table 3.8 API Functions: [Timer RG]

API Function Name	Function
R_TMR_RG0_Create	Performs initialization necessary to control the 16-bit timer RG0.
R_TMR_RG0_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer RG0.
r_tmr_rg0_interrupt	Performs processing in response to the timer interrupt.
R_TMR_RG0_Start	Starts the count for 16-bit timer RG0.
R_TMR_RG0_Stop	Ends the count for 16-bit timer RG0.
R_TMR_RG0_Set_PowerOff	Halts the clock supplied to the 16-bit timer RG0.
R_TMR_RG0_Get_PulseWidth	Reads the pulse width of the 16-bit timer RG0.

R_TMR_RG0_Create

Performs initialization necessary to control the 16-bit timer RG0.

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

[Syntax]

```
void        R_TMR_RG0_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RG0_Create_UserInit

Performs user-defined initialization relating to the 16-bit timer RG0.

Remark This API function is called as the R_TMR_RG0_Create callback routine.

[Syntax]

```
void    R_TMR_RG0_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_tmr_rg0_interrupt

Performs processing in response to the timer interrupt.

Remark This API function is called as the interrupt process corresponding to the timer interrupt.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_tmr_rg0_interrupt ( void );
```

CC-RL Compiler

```
static void     __near r_tmr_rg0_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RG0_Start

Starts the count for 16-bit timer RG0.

[Syntax]

```
void    R_TMR_RG0_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RG0_Stop

Ends the count for 16-bit timer RG0.

[Syntax]

```
void    R_TMR_RG0_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RG0_Set_PowerOff

Halts the clock supplied to the 16-bit timer RG0.

Remark Calling this API function changes the 16-bit timer RG0 to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_TMR_RG0_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_RG0_Get_PulseWidth

Reads the pulse width of the 16-bit timer RG0.

Remark 1. This API function can only be called when the 16-bit timer RG0 is being used for input capture function.

Remark 2. If there is an overflow (2 pulses or more) during pulse-width measurement, then the pulse width will not be read correctly.

[Syntax]

```
#include        "r_cg_macrodriver.h"
#include        "r_cg_timer.h"
MD_STATUS   R_TMR_RJ0_Get_PulseWidth ( uint32_t * const active_width,
 uint32_t * const inactive_width, timer_channel_t channel );
```

[Argument(s)]

I/O	Argument	Description
O	unit32_t * const *active_width*;	Pointer to an area storing the active level width that was read from the TRGIOA pin
O	uint32_t * const *inactive_width*;	Pointer to an area storing the inactive level width that was read from the TRGIOA pin
I	timer_channel_t *channel*;	Pin to read TMCHANNELA : TRGIOA0 pin TMCHANNELB : TRGIOB0 pin

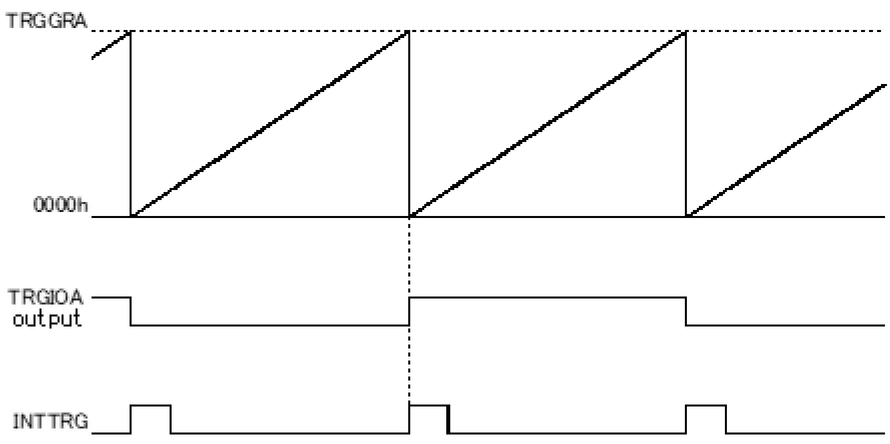
[Return value]

Macro	Description
MD_OK	Normal completion

Usage example (Output compare mode)

Perform toggle operation at fixed intervals and output a suquare wave with a duty factor of 50%.

[Waveform example]



[GUI setting example]

TimerRG				Used
	TMRG			Used
			Functions	Output compare function
			Count source setting	Internal clock
			Internal clock setting	Auto
			TRG counter setting (Counter clear)	Clear by TRGGRA compare match
			Register function setting (TRGGRC)	General register
			Register function setting (TRGGRD)	General register
			Compare value setting (TRGGRA)	100μs (Actual value：100)
			Output setting (TRGIOA pin)	Toggle output
			Enable TRGGRA compare match interrupt	Used
			Enable TRGoverflowinterrupt	Unused
			INTTRG priority	Low/ level3(low priority level)

[API setting example]

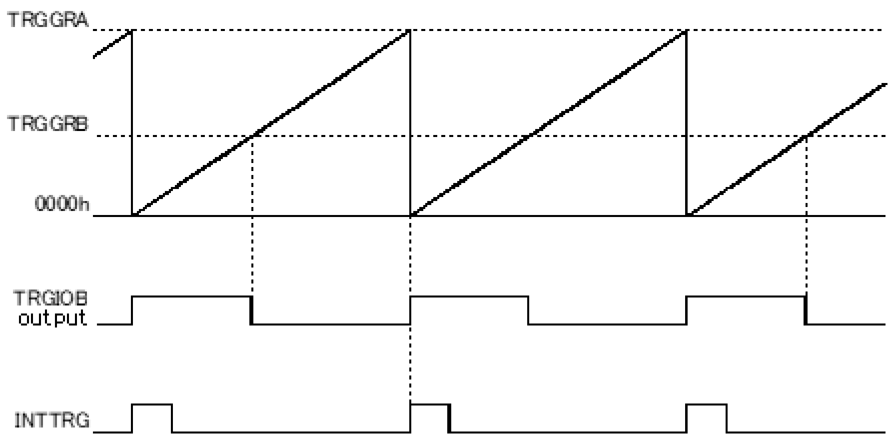
r_cg_main.c

```
void main(void)
{
        R_MAIN_UserInit();
        /* Start user code. Do not edit comment generated here */
        /* Start the TMRG module operation */
        R_TMRG0_Start();

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

Usage example (PWMmode)

Output PWM function with specified cycle and duty.

[Waveform example]



[GUI setting example]

TimerRG				Used
	TMRG			Used
			Functions	PWM mode
			Count source setting	Internal clock
			Internal clock setting	Auto
			Counter clear	Clear by TRGGRA compare match
			Register function setting (TRGGRC)	General register
			Register function setting (TRGGRD)	General register
			Cycle	100μs (Actual value：100)
			Duty	50(%) (Actual value：50)
			Enable TRGGRA compare match interrupt	Used
			Enable TRGGRB compare match interrupt	Unused
			Enable TRG overflowinterrup	Unused
			INTTRG priority	Low/ level3 (low priority level)

[API setting example]

r_cg_main.c

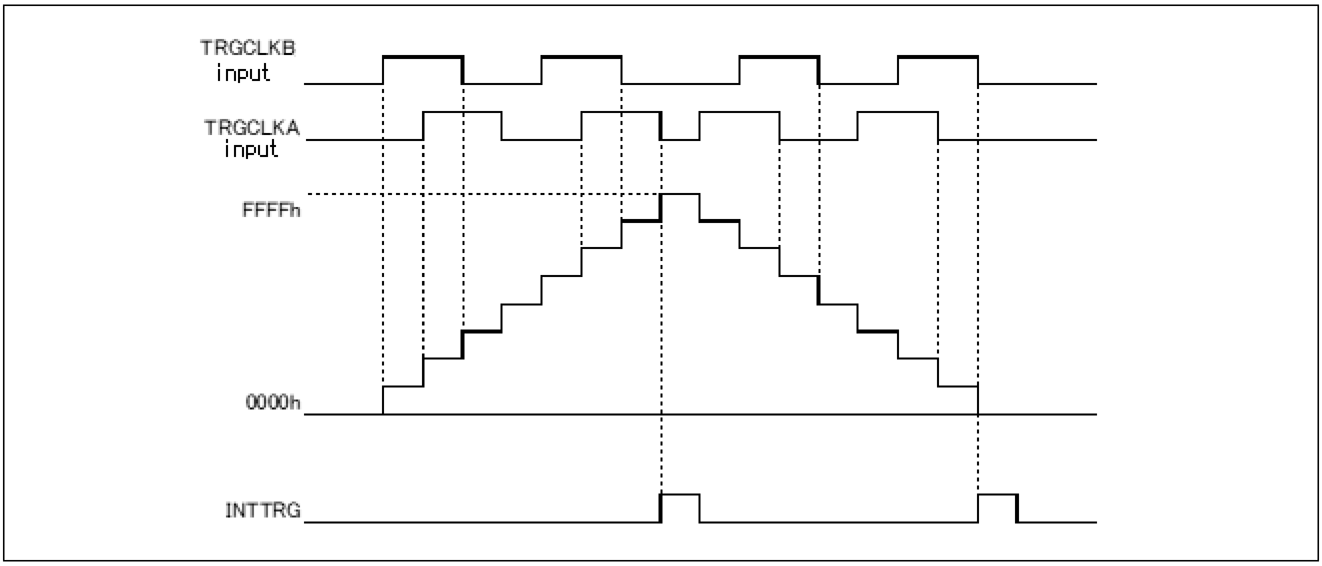
```
void main(void)
{
        R_MAIN_UserInit();
        /* Start user code. Do not edit comment generated here */
        /* Start the TMRG module operation */
        R_TMRG0_Start();

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

Usage example (Phase counting mode)

A phase difference between external input signals from two pins TRGCLKA and TRGCLKB is detected and the TRG register is incremented/decremented.

Count the number of TRG register overflow and underflow.

[Waveform example]



[GUI setting example]

TimerRG				Used
	TMRG			Used
			Functions	Phase counting mode
			Initial count	0
			Counter clear	Clear disabled
			CNTEN0	Used
			CNTEN1	Used
			CNTEN2	Used
			CNTEN3	Used
			CNTEN4	Used
			CNTEN5	Used
			CNTEN6	Used
			CNTEN7	Used
			Enable TRG overflow interrupt	Used
			Enable TRG under flowinterrupt	Used
			INTTRG priority	Low/ level3 (low priority level)

[API setting example]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start the TMRG module operation */
    R_TMRG0_Start();

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

r_cg_tmrg_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t inttrg_over_cnt = 0U;
volatile uint8_t inttrg_under_cnt = 0U;
/* End user code. Do not edit comment generated here */

static void __near r_tmrg0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    uint8_t temp_trg = 0U;

    /* === Count number of overflow or under flow === */
    /* Mask TRGSR resister to check overflow or underflow occurred */
    temp_trg = TRGSR & 0x0CU;

    if (temp_trg == 0x08U)
    {
        /* --- Count up number of overflow --- */
        inttrg_over_cnt++;

        /* --- Clear overflow Flag --- */
        TRGSR &= 0x07U;
    }
    else
    {
        /* --- Count up number of underflow --- */
        inttrg_under_cnt++;

        /* --- Clear under flow Flag --- */
        TRGSR &= 0x0BU;
    }
    /* End user code. Do not edit comment generated here */
}
```

3.3.9 Timer RX

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

Table 3.9 API Functions: [Timer RX]

API Function Name	Function
R_TMRX_Create	Performs initialization necessary to control the 16-bit timer RX.
R_TMRX_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer RX.
r_tmrx_interrupt	Performs processing in response to the timer interrupt.
R_TMRX_Start	Starts the count for 16-bit timer RX.
R_TMRX_Stop	Ends the count for 16-bit timer RX.
R_TMRX_Set_PowerOff	Halts the clock supplied to the 16-bit timer RX.
R_TMRX_Get_BufferValue	Reads the buffer value of TRX resister(16-bit timer RX).

R_TMRX_Create

Performs initialization necessary to control the 16-bit timer RX.

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

[Syntax]

```
void          R_TMRX_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMRX_Create_UserInit

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

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

[Syntax]

```
void R_TMRX_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_tmrx_interrupt

Performs processing in response to the timer interrupt.

Remark This API function is called as the interrupt process corresponding to the timer interrupt.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_tmrx_interrupt ( void );
```

CC-RL Compiler

```
static void     __near r_tmrx_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMRX_Start

Starts the count for 16-bit timer RX.

[Syntax]

```
void    R_TMRX_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMRX_Stop

Ends the count for 16-bit timer RX.

[Syntax]

```
void    R_TMRX_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMRX_Set_PowerOff

Halts the clock supplied to the 16-bit timer RX.

Remark Calling this API function changes the 16-bit timer RX to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_TMR_RX_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMRX_Get_BufferValue

Reads the buffer value of TRX register (16-bit timer RX).

[Syntax]

```
#include   "r_cg_macrodriver.h"
void       R_TMRX_Get_BufferValue ( uint32_t * const value );
```

[Argument(s)]

I/O	Argument	Description
O	unit32_t * const *value*;	Pointer to an area storing the buffer resister value of TRX resister

[Return value]

None.

Usage example

Stop the timer when TRX register overflows.

[GUI setting example]

TimerRX				Used
	TMRX			Used
			Timer RX operation setting	Used
			Clock setting	fCLK
			Count start source setting	Software
			Software reset enable signal setting	Enables software to reset counting
			Comparator 1 trigger setting	Transfer timer RX counter value to timer RX count buffer register. Set 0000H to timer RX count value, and continue counting
			Enable TRX overflow interrupt (INTTRX)	Used
			Priority	Low/ level3 (low priority level)

[API setting example]

r_cg_main.c

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

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

r_cg_tmrx_user.c

```
static void __near r_tmrx_interrupt(void)
{
	/* Start user code. Do not edit comment generated here */
	/* Stop TMRX counter */
	R_TMRX_Stop();
	/* End user code. Do not edit comment generated here */
}
```

3.3.10 16-bit timer KB

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

Table 3.10 API Functions: [16-bit Timers KB]

API Function Name	Function
R_TMR_KB_Create	Performs initialization necessary to control the 16-bit timer KB.
R_TMR_KB_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer KB.
r_tmr_kbm_interrupt	Performs processing in response to the timer interrupt.
R_TMR_KBm_Start	Starts the count for 16-bit timer KB.
R_TMR_KBm_Stop	Ends the count for 16-bit timer KB.
R_TMR_KBm_Set_PowerOff	Halts the clock supplied to the 16-bit timer KB.
R_TMR_KBmn_ForcedOutput_Start	Enables input of the trigger signal used for the forced output stop function.
R_TMR_KBmn_ForcedOutput_Stop	Disables input of the trigger signal used for the forced output stop function.
R_TMR_KBm_BatchOverwriteRequestOn	Enables batch overwriting of the compare register.
R_TMR_KBm_ForcedOutput_mn_Start	Enables input of the trigger signal used for the forced output stop function.
R_TMR_KBm_ForcedOutput_mn_Stop	Disables input of the trigger signal used for the forced output stop function.
R_TMR_KBm_Reset	Reset the 16-bit timer KB.

R_TMR_KB_Create

Performs initialization necessary to control the 16-bit timers KB.

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

[Syntax]

```
void        R_TMR_KB_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_KB_Create_UserInit

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

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

[Syntax]

```
void        R_TMR_KB_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_tmr_kb*m*_interrupt

Performs processing in response to the timer interrupt.

Remark This API function is called as the interrupt process corresponding to the timer interrupt.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_tmr_kbm_interrupt ( void );
```

CC-RL Compiler

```
static void    __near r_tmr_kbm_interrupt ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_KB*m*_Start

Starts the count for 16-bit timer KB.

[Syntax]

```
void    R_TMR_KBm_Start ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_KB*m*_Stop

Ends the count for 16-bit timer KB.

[Syntax]

```
void    R_TMR_KBm_Stop ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_KB*m*_Set_PowerOff

Halts the clock supplied to the 16-bit timer KB.

Remark Calling this API function changes the 16-bit timer KB to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_TMR_KBm_Set_PowerOff ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_KB*mn*_ForcedOutput_Start

Enables input of the trigger signal used for the forced output stop function.

[Syntax]

```
void    R_TMR_KBmn_ForcedOurput_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_KB*mn*_ForcedOutput_Stop

Disables input of the trigger signal used for the forced output stop function.

[Syntax]

```
void    R_TMR_KBmn_ForcedOutput_Stop ( void );
```

Remark   *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_KB*m*_BatchOverwriteRequestOn

Enables batch overwriting of the compare register.

Remark The timing for batch-overwriting the content of the compare register is when a count value and a value set in the compare register are matched or an external trigger is generated after calling this API function.

[Syntax]

```
void        R_TMR_KBm_BatchOverwritRequestOn ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_KB*m*_ForcedOutput_*mn*_Start

Enables input of the trigger signal used for the forced output stop function.

[Syntax]

```
void    R_TMR_KBm_ForcedOurput_mn_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_KB*m*_ForcedOutput_*mn*_Stop

Disables input of the trigger signal used for the forced output stop function.

[Syntax]

```
void    R_TMR_KBm_ForcedOutput_mn_Stop ( void );
```

Remark   *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_TMR_KB*m*_Reset

Reset the 16-bit timers KB.

[Syntax]

```
void        R_TMR_KBm_Reset ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

Usage example

Use timer as One-shot timer.

[GUI setting example]

Timer				Used
	TMKB			Used
			A/D trigger setting	Timer KB0 trigger source
		TMKB0		
			TMKB0	Standalone mode
		TMKB_STANDALONE_0		
			TKBO00	Unused
			TKBO01	Unused
			Cycle value	50ms (Actual value : 50)
			TKBO00 duty	0(%) (Actual value : 0%)
			TKBO01 duty	0(%) (Actual value : 0%)
			TKBO01delay	0μs (Actual value : 0)
			Use trigger input	Unused
			Use output gate function by TKC00	Unused
			Use output gate function by TKC01	Unused
			A/D conversion starttiming setting	0μs (Actual value : 0)
			End of timer channel 0 count, generate an interrupt (INTTMKB0)	Used
			Priority (INTTMKB0)	Low
			Smooth start function setting TKBO00	Unused
			Smooth start function setting TKBO01	Unused
			Dithering function setting TKBO00	Unused
			Dithering function setting TKBO01	Unused
			Forced output stop function setting TKBO00	Unused
			Forced output stop function setting TKBO01	Unused

[API setting example]

r_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start TMKB0 counter */
    R_TMR_KB0_Start();

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

r_cg_timer_user.c

```
static void __near r_tmr_kb0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop TMKB0 counter */
    R_TMR_KB0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

3.3.11 16-bit timer KC0

Below is a list of API functions output by the Code Generator for 16-bit timer KC0 use.

Table 3.11 API Functions: [16-bit Timer KC0]

API Function Name	Function
R_TMR_KC0_Create	Performs initialization necessary to control the 16-bit timer KC0.
R_TMR_KC0_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer KC0.
r_tmr_kc0_interrupt	Performs processing in response to the timer interrupt.
R_TMR_KC0_Start	Starts the count for 16-bit timer KC0.
R_TMR_KC0_Stop	Ends the count for 16-bit timer KC0.
R_TMR_KC0_Set_PowerOff	Halts the clock supplied to the 16-bit timer KC0.

R_TMR_KC0_Create

Performs initialization necessary to control the the 16-bit timer KC0.

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

[Syntax]

```
void        R_TMR_KC0_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_KC0_Create_UserInit

Performs user-defined initialization relating to the 16-bit timer KC0.

Remark This API function is called as the R_TMR_KC0_Create callback routine.

[Syntax]

```
void R_TMR_KC0_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_tmr_kc0_interrupt

Performs processing in response to the timer interrupt.

Remark This API function is called as the interrupt process corresponding to the timer interrupt.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_tmr_kc0_interrupt ( void );
```

CC-RL Compiler

```
static void     __near r_tmr_kc0_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_KC0_Start

Starts the count for 16-bit timer KC0.

[Syntax]

```
void    R_TMR_KC0_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_KC0_Stop

Ends the count for 16-bit timer KC0.

[Syntax]

```
void    R_TMR_KC0_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMR_KC0_Set_PowerOff

Halts the clock supplied to the 16-bit timer KC0.

Remark Calling this API function changes the 16-bit timer KC0 to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_TMR_KC0_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

Usage example

Use timer as One-shot timer.

[GUI setting example]

Timer				Used
	TMKC			Used
			Operation mode setting	Standalone mode
		TMKC0		
			TKCO00	Unused
			TKCO01	Unused
			TKCO02	Unused
			TKCO03	Unused
			TKCO04	Unused
			TKCO05	Unused
			Cycle value	50(Actual value：50)
			TKCO00 duty	0(%)(Actual value：0%)
			TKCO01 duty	0(%)(Actual value：0%)
			TKCO02 duty	0(%)(Actual value：0%)
			TKCO03 duty	0(%)(Actual value：0%)
			TKCO04 duty	0(%)(Actual value：0%)
			TKCO05 duty	0(%)(Actual value：0%)
			End of timer channel 0 count, generate an interrupt (INTTMKC0)	Used
			Priority (INTTMKC0)	Low

[API setting example]

r_main.c

```
void main(void)
{
        R_MAIN_UserInit();
        /* Start user code. Do not edit comment generated here */
        /* Start TMKC channel 0 counter */
        R_TMR_KC0_Start();

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

r_cg_timer_user.c

```
static void __near r_tmr_kc0_interrupt(void)
{
        /* Start user code. Do not edit comment generated here */
        /* Stop TMKC channel 0 counter */
        R_TMR_KC0_Stop();
        /* End user code. Do not edit comment generated here */
}
```

3.3.12 16-bit timer KB2

Below is a list of API functions output by the Code Generator for 16-bit timer KB2 use.

Table 3.12 API Functions: [16-bit Timer KB2]

API Function Name	Function
R_KB2m_Create	Performs initialization necessary to control the 16-bit timer KB2.
R_KB2m_Create_UserInit	Performs user-defined initialization relating to the 16-bit timer KB2.
r_kb2m_interrupt	Performs processing in response to the timer interrupt INTTKB2*m*.
R_KB2m_Start	Starts the count for 16-bit timer KB2.
R_KB2m_Stop	Ends the count for 16-bit timer KB2.
R_KB2m_Set_PowerOff	Halts the clock supplied to the 16-bit timer KB2.
R_KB2m_Simultaneous_Start	Starts the simultaneous start/stop mode.
R_KB2m_Simultaneous_Stop	Ends the simultaneous start/stop mode.
R_KB2m_Synchronous_Start	Starts the timer start/clear mode.
R_KB2m_Synchronous_Stop	Ends the timer start/clear mode.
R_KB2m_TKBOn0_Forced_Output_Stop_Function1_Start	Starts forced output stop function 1 for timer output TKBO*n*0.
R_KB2m_TKBOn0_Forced_Output_Stop_Function1_Stop	Ends forced output stop function 1 for timer output TKBO*n*0.
R_KB2m_TKBOn1_Forced_Output_Stop_Function1_Start	Starts forced output stop function 2 for timer output TKBO*n*1.
R_KB2m_TKBOn1_Forced_Output_Stop_Function1_Stop	Starts forced output stop function 2 for timer output TKBO*n*1.
R_KB2m_TKBOn0_DitheringFunction_Start	Starts dithering function for timer output TKBO*n*0.
R_KB2m_TKBOn0_DitheringFunction_Stop	Ends dithering function for timer output TKBO*n*0.
R_KB2m_TKBOn1_DitheringFunction_Start	Starts dithering function for timer output TKBO*n*1.
R_KB2m_TKBOn1_DitheringFunction_Stop	Ends dithering function for timer output TKBO*n*1.
R_KB2m_TKBOn0_SmoothStartFunction_Start	Starts smooth start function for timer output TKBO*n*0.
R_KB2m_TKBOn0_SmoothStartFunction_Stop	Ends smooth start function for timer output TKBO*n*0.
R_KB2m_TKBOn1_SmoothStartFunction_Start	Starts smooth start function for timer output TKBO*n*1.
R_KB2m_TKBOn1_SmoothStartFunction_Stop	Ends smooth start function for timer output TKBO*n*1.
R_KB2m_BatchOverwriteRequestOn	Enables batch overwriting of the compare register.

R_KB2*m*_Create

Performs initialization necessary to control the 16-bit timer KB2.

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

[Syntax]

```
void        R_KB2m_Create ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_Create_UserInit

Performs user-defined initialization relating to the 16-bit timer KB2.

Remark This API function is called as the R_KB2m_Create callback routine.

[Syntax]

```
void    R_KB2m_Create_UserInit ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_kb2*m*_interrupt

Performs processing in response to the timer interrupt INTTKB2*m*.

Remark This API function is called as the interrupt process corresponding to the timer interrupt INTTKB2*m*.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_kb2m_interrupt ( void );
```

CC-RL Compiler

```
static void     __near r_kb2m_interrupt ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_Start

Starts the count for 16-bit timer KB2.

[Syntax]

```
void    R_KB2m_Start ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_Stop

Ends the count for 16-bit timer KB2.

[Syntax]

```
void    R_KB2m_Stop ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_Set_PowerOff

Halts the clock supplied to the 16-bit timer KB2.

[Syntax]

```
void    R_KB2m_Set_PowerOff ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_Simultaneous_Start

Starts the simultaneous start/stop mode.

[Syntax]

```
void    R_KB2m_Simultaneous_Start ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_Simultaneous_Stop

Ends the simultaneous start/stop mode.

[Syntax]

```
void    R_KB2m_Simultaneous_Stop ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_Synchronous_Start

Starts the timer start/clear mode.

[Syntax]

```
void    R_KB2m_Synchronous_Start ( void );
```

Remark   *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_Synchronous_Stop

Ends the timer start/clear mode.

[Syntax]

```
void    R_KB2m_Synchronous_Stop ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_TKBO*n*0_Forced_Output_Stop_Function1_Start

Starts forced output stop function 1 for timer output TKBO*n*0.

[Syntax]

```
void    R_KB2m_TKBOn0_Forced_Ourput_Stop_Function1_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_TKBO*n*0_Forced_Output_Stop_Function1_Stop

Ends forced output stop function 1 for timer output TKBO*n*0.

[Syntax]

```
void    R_KB2m_TKBOn0_Forced_Output_Stop_Function1_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_TKBO*n*1_Forced_Output_Stop_Function1_Start

Starts forced output stop function 2 for timer output TKBO*n*1.

[Syntax]

```
void    R_KB2m_TKBOn1_Forced_Ourput_Stop_Function1_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_TKBO*n*1_Forced_Output_Stop_Function1_Stop

Ends forced output stop function 2 for timer output TKBO*n*1.

[Syntax]

```
void    R_KB2m_TKBOn1_Forced_Output_Stop_Function1_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_TKBO*n*0_DitheringFunction_Start

Starts dithering function for timer output TKBO*n*0.

[Syntax]

```
void    R_KB2m_TKBOn0_DitheringFunction_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_TKBO*n*0_DitheringFunction_Stop

Ends dithering function for timer output TKBO*n*0.

[Syntax]

```
void    R_KB2m_TKBOn0_DitheringFunction_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_TKBO*n*1_DitheringFunction_Start

Starts dithering function for timer output TKBO*n*1.

[Syntax]

```
void    R_KB2m_TKBOn1_DitheringFunction_Start ( void );
```

Remark   *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_TKBO*n*1_DitheringFunction_Stop

Ends dithering function for timer output TKBO*n*1.

[Syntax]

```
void    R_KB2m_TKBOn1_DitheringFunction_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_TKBO*n*0_SmoothStartFunction_Start

Starts smooth start function for timer output TKBO*n*0.

[Syntax]

```
void    R_KB2m_TKBOn0_SmoothStartFunction_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_TKBO*n*0_SmoothStartFunction_Stop

Ends smooth start function for timer output TKBO*n*0.

[Syntax]

```
void    R_KB2m_TKBOn0_SmoothStartFunction_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_TKBO*n*1_SmoothStartFunction_Start

Starts smooth start function for timer output TKBO*n*1.

[Syntax]

```
void    R_KB2m_TKBOn1_SmoothStartFunction_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_TKBO*n*1_SmoothStartFunction_Stop

Ends smooth start function for timer output TKBO*n*1.

[Syntax]

```
void    R_KB2m_TKBOn1_SmoothStartFunction_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_KB2*m*_BatchOverwriteRequestOn

Enables batch overwriting of the compare register.

Remark The timing for batch-overwriting the content of the compare register is when a count value and a value set in the compare register are matched or an external trigger is generated after calling this API function.

[Syntax]

```
void        R_KB2m_BatchOverwritRequestOn ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

Usage example

Use timer as One-shot timer.

[GUI setting example]

TimerKB2				Used
	KB2			Used
		TKB20		Used
			TKB20	Standalone mode (period controlled by TKBCR00）
			Clock setting	TKBTCK0 selected
			Operation clock setting	fCLK
			Pulse period	1ms(Actual value：1)
			Duty (TKBO00 output)	0(%)(Actual value：0)
			Duty (TKBO01 output)	0(%)(Actual value：0)
			Delay (TKBO01 output)	0(%)(Actual value：0)
			PWM output smooth start function of TKBO00 setting	Unused
			PWM output smooth start function of TKBO01 setting	Unused
			PWM output dithering function of TKBO00 setting	Unused
			PWM output dithering function of TKBO01 setting	Unused
			TKBTGCR0 value	0
			TKBO00 output setting	Disabled
			TKBO01 output setting	Disabled
			End of timer KB20 count interrupt (INTTKB20)	Used
			Priority	Low
			Forced output stop function setting (TKBO00)	Disabled
			Forced output stop function setting (TKBO01)	Disabled

[API setting example]

r_cg_main.c

```
void main(void)
{
        R_MAIN_UserInit();
        /* Start user code. Do not edit comment generated here */
        /* Start KB20 module operation */
        R_KB20_Start();

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

r_cg_kb2_user.c

```
static void __near r_kb20_interrupt(void)
{
        /* Start user code. Do not edit comment generated here */
        /* Stop KB20 module operation */
        R_KB20_Stop();
        /* End user code. Do not edit comment generated here */
}
```

3.3.13 Real-time clock

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

Table 3.13 API Functions: [Real-time Clock] (1)

API Function Name	Function
R_RTC_Create	Performs initialization necessary to control the real-time clock.
R_RTC_Create_UserInit	Performs user-defined initialization relating to the real-time clock.
r_rtc_interrupt	Performs processing in response to the real-time clock interrupt INTRTC.
R_RTC_Start	Starts the count of the real-time clock (year, month, weekday, day, hour, minute, second).
R_RTC_Stop	Ends the count of the real-time clock (year, month, weekday, day, hour, minute, second).
R_RTC_Set_PowerOff	Halts the clock supplied to the real-time clock.
R_RTC_Set_HourSystem	Sets the clock type (12-hour or 24-hour clock) of the real-time clock.
R_RTC_Set_CounterValue	Sets the counter value of the real-time clock.
R_RTC_Set_CalendarCounterValue	Sets the counter value of the real-time clock.(in the case of the calendar mode setting)
R_RTC_Set_BinaryCounterValue	Sets the counter value of the real-time clock.(in the case of the binary mode setting)
R_RTC_Get_CounterValue	Reads the counter value of the real-time clock.
R_RTC_Get_CalendarCounterValue	Reads the counter value of the real-time clock.(in the case of the calendar mode setting)
R_RTC_Get_BinaryCounterValue	Reads the counter value of the real-time clock.(in the case of the binary mode setting)
R_RTC_Set_ConstPeriodInterruptOn	Sets the cycle of the interrupts INTRTC, then starts the cyclic interrupt function.
R_RTC_Set_ConstPeriodInterruptOff	Ends the cyclic interrupt function.
r_rtc_callback_constperiod	Performs processing in response to the cyclic interrupt INTRTC.
R_RTC_Set_AlarmOn	Starts the alarm interrupt function.
R_RTC_Set_CalendarAlarmOn	Starts the alarm interrupt function.(in the case of the calendar mode setting)
R_RTC_Set_BinaryAlarmOn	Starts the alarm interrupt function.(in the case of the binary mode setting)
R_RTC_Set_AlarmOff	Ends the alarm interrupt function.
R_RTC_Set_AlarmValue	Sets the alarm conditions (weekday, hour, minute).
R_RTC_Get_CalenderAlarmValue	Sets the alarm conditions (year, month, weekday, day, hour, minute, second).(in the case of the calendar mode setting)
R_RTC_Set_BinaryAlarmValue	Sets the alarm conditions.(in the case of the binary mode setting)
R_RTC_Get_AlarmValue	Reads the alarm conditions (weekday, hour, minute).

Table 3.14 API Functions: [Real-time Clock] (2)

API Function Name	Function
R_RTC_Get_CalenderAlarmValue	Reads the alarm conditions (year, month, weekday, day, hour, minute, second).(in the case of the calendar mode setting)
R_RTC_Get_BinaryAlarmValue	Reads the alarm conditions.(in the case of the binary mode setting)
r_rtc_callback_alarm	Performs processing in response to the alarm interrupt INTRTC.
R_RTC_Set_RTC1HZOn	Enables output of the correction clock (1 Hz) to the RTC1HZ pin.
R_RTC_Set_RTC1HZOff	Disables output of the correction clock (1 Hz) to the RTC1HZ pin.
R_RTC_Set_RTCOUTOn	Enables output of the RTCOUT.
R_RTC_Set_RTCOUTOff	Disables output of the RTCOUT.
r_rtc_alarminterrupt	Performs processing in response to the alarm interrupt INTRTCALM.
r_rtc_periodinterrupt	Performs processing in response to the periodic interrupt INTRTCPRD.
r_rtc_callback_periodic	Performs processing in response to the cyclic interrupt INTRTC.

R_RTC_Create

Performs initialization necessary to control the real-time 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 real-time 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_interrupt

Performs processing in response to the real-time clock interrupt INTRTC.

Remark This API function is called as the interrupt process corresponding to the real-time clock interrupt INTRTC.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_rtc_interrupt ( void );
```

CC-RL Compiler

```
Static void    __near r_rtc_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Start

Starts the count of the real-time clock (year, month, weekday, day, hour, minute, second).

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_RTC_Stop

Ends the count of the real-time clock (year, month, weekday, day, hour, minute, second).

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_PowerOff

Halts the clock supplied to the real-time clock.

Remark 1. Calling this API function changes the real-time clock to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

Remark 2. This API function stops the clock supply to the real-time clock, by operating the RTCEN bit of peripheral enable register *n*.
For this reason, this API function also stops the clock supply to other peripheral devices sharing the RTCEN bit (e.g. interval timer).

[Syntax]

```
void        R_RTC_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_HourSystem

Sets the clock type (12-hour or 24-hour clock) of the real-time clock.

[Syntax]

```
#include        "r_cg_macrodriver.h"
#include        "r_cg_rtc.h"
MD_STATUS   R_RTC_Set_HourSystem ( rtc_hour_system_t hour_system );
```

[Argument(s)]

I/O	Argument	Description
I	rtc_hour_system_t *hour_system*;	Clock type HOUR12 : 12 -hour clock HOUR24 : 24 -hour clock

[Return value]

Macro	Description
MD_OK	Normal completion
MD_BUSY1	Executing count process (before change to setting)
MD_BUSY2	Stopping count process (after change to setting)
MD_ARGERROR	Invalid argument specification

Remark If MD_BUSY1 or MD_BUSY2 is returned, it may be because the counter-operation is stopped, or the counter operation start wait time is too short, so make the value of the RTC_WAITTIME macro defined in the header file "r_cg_rtc.h" larger.

R_RTC_Set_CounterValue

Sets the counter value (year, month, weekday, day, hour, minute, second) of the real-time clock.

Remark To rewrite the SEC, MIN, HOUR, WEEK, DAY, MONTH, and YEAR registers with this function while the counter is operating (RTCE = 1), disable the interrupt processing of INTRTC with the interrupt mask flag register and then call.

[Syntax]

```
#include     "r_cg_macrodriver.h"
#include     "r_cg_rtc.h"
MD_STATUS   R_RTC_Set_CounterValue ( rtc_counter_value_t counter_write_val );
```

[Argument(s)]

I/O	Argument	Description
I	rtc_counter_value_t *counter_write_val*;	Counter value

Remark Below is an example of the structure rtc_counter_value_t (counter value) for the real-time clock.

```
typedef struct {
        uint8_t sec;       /* Second */
        uint8_t min;       /* Minute */
        uint8_t hour;      /* Hour */
        uint8_t day;       /* Day */
        uint8_t week;      /* Weekday (0: Sunday, 6: Saturday) */
        uint8_t month;     /* Month */
        uint16_t year;     /* Year */
} rtc_counter_value_t;
```

[Return value]

Macro	Description
MD_OK	Normal completion
MD_BUSY1	Executing count process (before change to setting)
MD_BUSY2	Stopping count process (after change to setting)

Remark If MD_BUSY1 or MD_BUSY2 is returned, it may be because the counter-operation is stopped, or the counter operation start wait time is too short, so make the value of the RTC_WAITTIME macro defined in the header file "r_cg_rtc.h" larger.

R_RTC_Set_CalendarCounterValue

Sets the counter value of the real-time clock.(in the case of the calendar mode setting)

[Syntax]

```
#include        "r_cg_macrodriver.h"
#include        "r_cg_rtc.h"
MD_STATUS   R_RTC_Set_CalendarCounterValue ( rtc_counter_value_t counter_write_val );
```

[Argument(s)]

I/O	Argument	Description
I	rtc_counter_value_t *counter_write_val*;	Counter value

Remark See R_RTC_Set_CounterValue for details about the rtc_counter_value_t counter value.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_BUSY1	Executing count process (before change to setting)

Remark If MD_BUSY1 is returned, it may be because the counter operation start wait time is too short, so make the value of the RTC_WAITTIME macro defined in the header file "r_cg_rtc.h" larger.

R_RTC_Set_BinaryCounterValue

Sets the counter value of the real-time clock.(in the case of the binary mode setting)

[Syntax]

```
#include        "r_cg_macrodriver.h"
#include        "r_cg_rtc.h"
MD_STATUS   R_RTC_Set_BinaryCounterValue ( uint32_t counter_write_val );
```

[Argument(s)]

I/O	Argument	Description
I	uint32_t *counter_write_val*;	Counter value

[Return value]

Macro	Description
MD_OK	Normal completion
MD_BUSY1	Executing count process (before change to setting)

Remark If MD_BUSY1 is returned, it may be because the counter operation start wait time is too short, so make the value of the RTC_WAITTIME macro defined in the header file "r_cg_rtc.h" larger.

R_RTC_Get_CounterValue

Reads the counter value (year, month, weekday, day, hour, minute, second) of the real-time clock.

[Syntax]

```
#include        "r_cg_macrodriver.h"
#include        "r_cg_rtc.h"
MD_STATUS   R_RTC_Get_CounterValue ( rtc_counter_value_t * const counter_read_val );
```

[Argument(s)]

I/O	Argument	Description
O	rtc_counter_value_t * const *counter_read_val*;	Pointer to structure in which to store the counter value being read

Remark See R_RTC_Set_CounterValue for details about the rtc_counter_value_t counter value.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_BUSY1	Executing count process (before reading)
MD_BUSY2	Stopping count process (after reading)

Remark If MD_BUSY1 or MD_BUSY2 is returned, it may be because the counter-operation is stopped, or the counter operation start wait time is too short, so make the value of the RTC_WAITTIME macro defined in the header file "r_cg_rtc.h" larger.

R_RTC_Get_CalendarCounterValue

Reads the counter value of the real-time clock.(in the case of the calendar mode setting)

[Syntax]

```
#include        "r_cg_macrodriver.h"
#include        "r_cg_rtc.h"
MD_STATUS   R_RTC_Get_CalendarCounterValue ( rtc_counter_value_t * const counter_read_val );
```

[Argument(s)]

I/O	Argument	Description
O	rtc_counter_value_t * const *counter_read_val*;	Pointer to structure in which to store the counter value being read

Remark See R_RTC_Set_CounterValue for details about the rtc_counter_value_t counter value.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR	Read failure

R_RTC_Get_BinaryCounterValue

Reads the counter value of the real-time clock.(in the case of the calendar mode setting)

[Syntax]

```
#include        "r_cg_macrodriver.h"
#include        "r_cg_rtc.h"
MD_STATUS   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 structure in which to store the counter value being read

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR	Read failure

R_RTC_Set_ConstPeriodInterruptOn

Sets the cycle of the interrupts INTRTC, then starts the cyclic interrupt function.

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	rtc_int_period_t *period*;	Interrupt INTRTC cycle HALFSEC : 0.5 seconds ONESEC : 1 second ONEMIN : 1 minute ONEHOUR : 1 hour ONEDAY : 1 day ONEMONTH : 1 month

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_RTC_Set_ConstPeriodInterruptOff

Ends the cyclic interrupt function.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

r_rtc_callback_constperiod

Performs processing in response to the cyclic interrupt INTRTC.

Remark This API function is called as the callback routine of interrupt process r_rtc_interrupt corresponding to the cyclic interrupt INTRTC.

[Syntax]

```
static void     r_rtc_callback_constperiod ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_AlarmOn

Starts the alarm interrupt function.

[Syntax]

```
void    R_RTC_Set_AlarmOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_CalendarAlarmOn

Starts the alarm interrupt function.(in the case of the calendar mode setting)

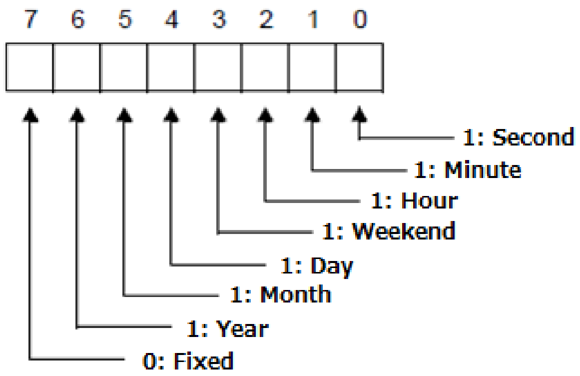
[Syntax]

```
#include   "r_cg_macrodriver.h"
#include   "r_cg_rtc.h"
void       R_RTC_Set_CalendarAlarmOn ( uint8_t enb_set );
```

[Argument(s)]

I/O	Argument	Description
I	uint8_t *enb_set*;	Alarm enable

Remark Below is shown the structure enb_set.



[Return value]

None.

R_RTC_Set_BinaryAlarmOn

Starts the alarm interrupt function.(in the case of the binary mode setting)

[Syntax]

```
void    R_RTC_Set_BinaryAlarm ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_AlarmOff

Ends the alarm interrupt function.

[Syntax]

```
void    R_RTC_Set_AlarmOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_AlarmValue

Sets the alarm conditions (weekday, hour, minute).

[Syntax]

```
#include   "r_cg_macrodriver.h"
#include   "r_cg_rtc.h"
void       R_RTC_Set_AlarmValue ( rtc_alarm_value_t alarm_val );
```

[Argument(s)]

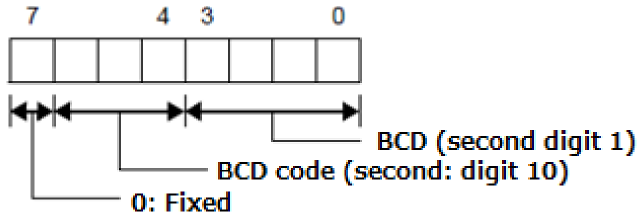
I/O	Argument	Description
I	rtc_alarm_value_t *alarm_val*;	Alarm conditions (weekday, hour, minute)

Remark Below is shown the structure rtc_alarm_value_t (alarm conditions).(The structure is different according to the device.)

```
typedef struct {
        uint8_t sec;        /* Second */
        uint8_t min;        /* Minute */
        uint8_t hour;       /* Hour */
        uint8_t week;       /* Weekday (0: Sunday, 6: Saturday) */
        uint8_t day;        /* Day */
        uint8_t month;      /* Month */
        uint16_t year;      /* Year */
} rtc_alarm_value_t;
```

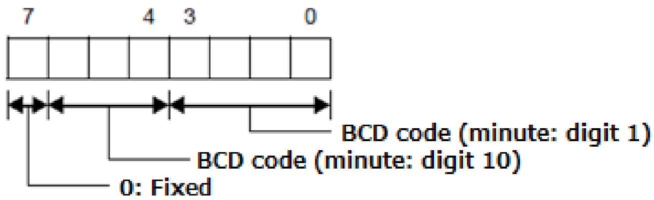
- alarmws (Second)

  Below are shown the meanings of each bit of the structure member alarmws.



- alarmwm (Minute)

  Below are shown the meanings of each bit of the structure member alarmwm.



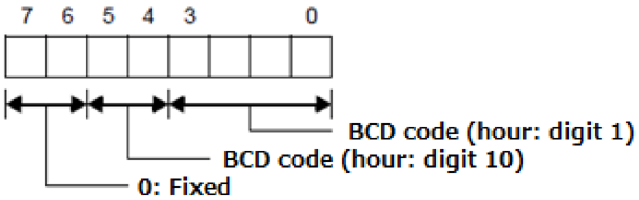
- alarmwh (Hour)

  Below are shown the meanings of each bit of the structure member alarmwh.

  If the real-time clock is set to the 12-hour clock, then bit 5 has the following meaning.

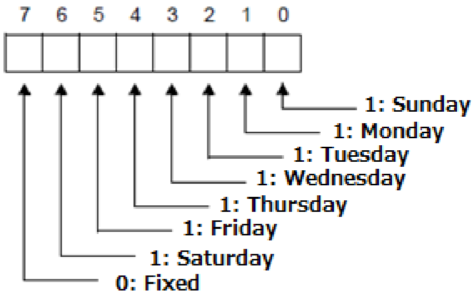
  0: AM

  1: PM



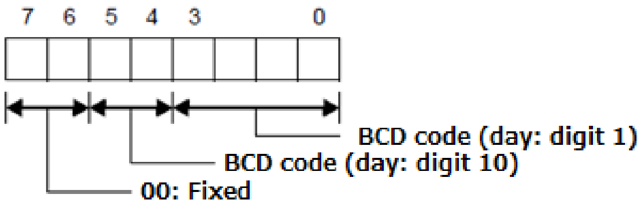
- alarmww (Weekday)

  Below are shown the meanings of each bit of the structure member alarmww.



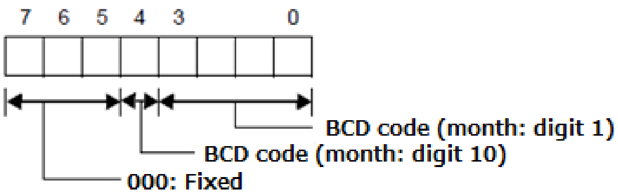
- alarmwd (Day)

  Below are shown the meanings of each bit of the structure member alarmwd.



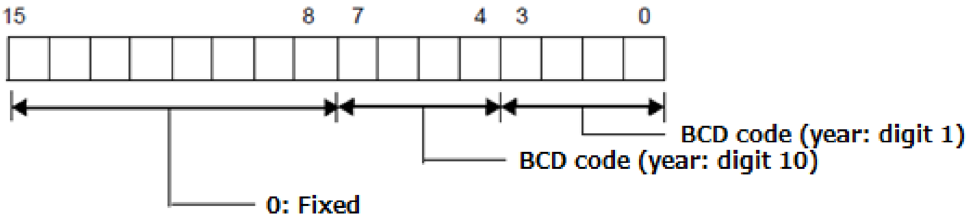
- alarmwmn (Month)

  Below are shown the meanings of each bit of the structure member alarmwmn.



- alarmwy (Year)

  Below are shown the meanings of each bit of the structure member alarmwmn.



[Return value]

None.

R_RTC_Set_CalenderAlarmValue

Sets the alarm conditions (year, month, weekday, day, hour, minute, second).(in the case of the calendar mode setting)

[Syntax]

```
#include   "r_cg_macrodriver.h"
#include   "r_cg_rtc.h"
void       R_RTC_Set_CalenderAlarmValue ( rtc_alarm_value_t alarm_val );
```

[Argument(s)]

I/O	Argument	Description
I	rtc_alarm_value_t *alarm_val*;	Alarm conditions (second, minute, hour, weekday, day, month, year)

Remark See R_RTC_Set_AlarmValue for details about rtc_alarm_value_t (alarm conditions).

[Return value]

None.

R_RTC_Set_BinaryAlarmValue

Sets the alarm conditions.(in the case of the binary mode setting)

[Syntax]

```
#include   "r_cg_macrodriver.h"
#include   "r_cg_rtc.h"
void       R_RTC_Set_BinaryAlarmValue ( uint32_t alarm_enable, uint32_t alarm_val );
```

[Argument(s)]

I/O	Argument	Description
I	uint32_t *alarm_enable*;	Alarm enable (Set the value to the Binary Counter Alarm Enable Register)
I	uint32_t *alarm_val*;	Alarm conditions (count value) (Set the value to the Binary Counter Alarm Register)

[Return value]

None.

R_RTC_Get_AlarmValue

Reads the alarm conditions (weekday, hour, minute).

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_rtc.h"
void        R_RTC_Get_AlarmValue ( rtc_alarm_value_t * const alarm_val );
```

[Argument(s)]

I/O	Argument	Description
O	rtc_alarm_value_t * const *alarm_val*;	Pointer to structure in which to store the conditions being read

Remark See R_RTC_Set_AlarmValue for details about rtc_alarm_value_t (alarm conditions).

[Return value]

None.

R_RTC_Get_CalenderAlarmValue

Reads the alarm conditions (year, month, weekday, day, hour, minute, second).(in the case of the calendar mode setting)

[Syntax]

```
#include   "r_cg_macrodriver.h"
#include   "r_cg_rtc.h"
void       R_RTC_Get_CalenderAlarmValue ( rtc_alarm_value_t * const alarm_val );
```

[Argument(s)]

I/O	Argument	Description
O	rtc_alarm_value_t * const *alarm_val*;	Pointer to structure in which to store the conditions being read

Remark See R_RTC_Set_AlarmValue for details about rtc_alarm_value_t (alarm conditions).

[Return value]

None.

R_RTC_Get_BinaryAlarmValue

Reads the alarm conditions (weekday, hour, minute).(in the case of the binary mode setting)

[Syntax]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_rtc.h"
void        R_RTC_Get_BinaryAlarmValue ( uint32_t * const alarm_enable,
 uint32_t * const alarm_val );
```

[Argument(s)]

I/O	Argument	Description
O	uint32_t * const *alarm_enable*;	Pointer to structure in which to store the alarm enable value being read
O	uint32_t * const *alarm_val*;	Pointer to structure in which to store the conditions being read

[Return value]

None.

r_rtc_callback_alarm

Performs processing in response to the alarm interrupt INTRTC.

Remark This API function is called as the callback routine of interrupt process r_rtc_interrupt corresponding to the alarm interrupt INTRTC.

[Syntax]

```
void          r_rtc_callback_alarm ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_RTC1HZOn

Enables output of the correction clock (1 Hz) to the RTC1HZ pin.

[Syntax]

```
void        R_RTC_Set_RTC1HZOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_RTC1HZOff

Disables output of the correction clock (1 Hz) to the RTC1HZ pin.

[Syntax]

```
void        R_RTC_Set_RTC1HZOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_RTCOUTOn

Enables output of the RTCOUT.

[Syntax]

```
void        R_RTC_Set_RTCOUTOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_RTC_Set_RTCOUTOff

Disables output of the RTCOUT.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

r_rtc_alarminterrupt

Performs processing in response to the alarm interrupt INTRTCALM.

Remark This API function is called as the interrupt process corresponding to the alarm interrupt INTRTCALM.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_rtc_alarminterrupt ( void );
```

CC-RL Compiler

```
static void     __near r_rtc_alarminterrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_rtc_periodinterrupt

Performs processing in response to the periodic interrupt INTRTCPRD.

Remark This API function is called as the interrupt process corresponding to the periodic interrupt INTRTCPRD.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_rtc_periodinterrupt ( void );
```

CC-RL Compiler

```
static void    __near r_rtc_periodinterrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_rtc_callback_periodic

Performs processing in response to the cyclic interrupt INTRTC.

Remark This API function is called as the callback routine of interrupt process r_rtc_interrupt corresponding to the cyclic interrupt INTRTC.

[Syntax]

```
static void     r_rtc_callback_periodic ( void );
```

[Argument(s)]

None.

[Return value]

None.

Usage example

Generate an alarm intetrrupt every 10 minutes between AM7:00 and AM7:59.

[GUI setting example]

Real-time clock				Used
	RTC			Used
			Real-time clock operation setting	Used
			Hour-system selection	24-hour
			Set real-time clock initial value	Used 04/01/2018 00:00:00
			Enable output of RTC1HZ pin (1 Hz)	Unused
			Use alarm detection function	Used
			Set alarm initial value	Used
			Week day	Sunday Monday Tuesday Wednesday Thursday Friday Saturday
			Hour: Minute	AM 07:00
			Used as alarm interrupt function (INTRTC)	Used
			Used as condtant-period interrupt function (INTRTC)	Unused
			Priority(INTRTC)	Low

[API setting example]

r_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Enable the real-time clock */
    R_RTC_Start();

    /* Start the alarm operation */
    R_RTC_Set_AlarmOn();

    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_alarm_value_t alarm_val;
/* End user code. Do not edit comment generated here */

static void r_rtc_callback_alarm(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Get alarm value */
    R_RTC_Get_AlarmValue((rtc_alarm_value_t *)&alarm_val);

    if ((alarm_val.alarmwm + 0x10U) <= 0x59U)
    {
        alarm_val.alarmwm += 0x10U;
        /* Set alarm value */
        R_RTC_Set_AlarmValue(alarm_val);
    }
    else
    {
        /* Stop the alarm operation */
        R_RTC_Set_AlarmOff();
    }
    /* End user code. Do not edit comment generated here */
}
```

3.3.14 Subsystem clock frequency measurement circuit

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

Table 3.15 API Functions: [Subsystem Clock Frequency Measurement Circuit]

API Function Name	Function
R_FMC_Create	Performs initialization necessary to control the subsystem clock frequency measurement circuit.
R_FMC_Create_UserInit	Performs user-defined initialization relating to the subsystem clock frequency measurement circuit.
r_fmc_interrupt	Performs processing in response to the end of frequency measurement interrupt INTFM.
R_FMC_Start	Starts measurement of the frequency that uses the subsystem clock frequency measurement circuit.
R_FMC_Stop	Ends measurement of the frequency that uses the subsystem clock frequency measurement circuit.
R_FMC_Set_PowerOff	Halts the clock supplied to the subsystem clock frequency measurement circuit.

R_FMC_Create

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

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

[Syntax]

```
void        R_FMC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_FMC_Create_UserInit

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

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

[Syntax]

```
void R_FMC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_fmc_interrupt

Performs processing in response to the end of frequency measurement interrupt INTFM.

Remark This API function is called as the interrupt process corresponding to the end of frequency measurement interrupt INTFM.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_fmc_interrupt ( void );
```

CC-RL Compiler

```
static void    __near r_fmc_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_FMC_Start

Starts measurement of the frequency that uses the subsystem clock frequency measurement circuit.

[Syntax]

```
void        R_FMC_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_FMC_Stop

Ends measurement of the frequency that uses the subsystem clock frequency measurement circuit.

[Syntax]

```
void        R_FMC_stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_FMC_Set_PowerOff

Halts the clock supplied to the subsystem clock frequency measurement circuit.

Remark Calling this API function changes the subsystem clock frequency measurement circuit to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_FMC_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.3.15 12-bit interval timer

Below is a list of API functions output by the Code Generator for 12-bit interval timer use.

Table 3.16 API Functions: [12-Bit Interval Timer]

API Function Name	Function
R_IT_Create	Performs initialization necessary to control the 12-bit interval timer.
R_IT_Create_UserInit	Performs user-defined initialization relating to the 12-bit interval timer.
r_it_interrupt	Performs processing in response to the 12-bit interval timer interrupt INTIT.
R_IT_Start	Starts the count of the 12-bit interval timer.
R_IT_Stop	Ends the count of the 12-bit interval timer.
R_IT_Reset	Reset the 12-bit interval timer.
R_IT_Set_PowerOff	Halts the clock supplied to the 12-bit interval timer.

R_IT_Create

Performs initialization necessary to control the 12-bit interval timer.

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

[Syntax]

```
void R_IT_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IT_Create_UserInit

Performs user-defined initialization relating to the 12-bit interval timer.

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

[Syntax]

```
void        R_IT_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_it_interrupt

Performs processing in response to the 12-bit interval timer interrupt INTIT.

Remark This API function is called as the interrupt process corresponding to the 12-bit interval timer interrupt INTIT.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_it_interrupt ( void );
```

CC-RL Compiler

```
static void    __near r_it_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IT_Start

Starts the count of the 12-bit interval timer.

Remark The timer is cleared when the count operation stops.

[Syntax]

```
void        R_IT_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IT_Stop

Ends the count of the 12-bit interval timer.

Remark The timer is cleared when the count operation stops.

[Syntax]

```
void        R_IT_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IT_Reset

Reset the 12-bit interval timer.

[Syntax]

```
void        R_IT_Reset ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_IT_Set_PowerOff

Halts the clock supplied to the 12-bit interval timer.

Remark 1. Calling this API function changes the 12-bit interval timer to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

Remark 2. This API function stops the clock supply to the 12-bit interval timer, by operating the RTCEN bit of peripheral enable register *n*.
For this reason, this API function also stops the clock supply to other peripheral devices sharing the RTCEN bit (e.g. real-timer clock).

[Syntax]

```
void        R_IT_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

Usage example

Use timer as One-shot timer.

[GUI setting example]

12 bit interval timer				Used
	IT			Used
			Interval timer operation setting	Used
			Interval value	100ms (Actual value：100)
			Detection of interval signal (INTIT)	Used
			Priority	Low

[API setting example]

r_cg_main.c

```
void main(void)
{
        R_MAIN_UserInit();
        /* Start user code. Do not edit comment generated here */
        /* Start IT module operation */
        R_IT_Start();

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

r_cg_it_user.c

```
static void __near r_it_interrupt(void)
{
        /* Start user code. Do not edit comment generated here */
        /* Stop IT module operation */
        R_IT_Stop();
        /* End user code. Do not edit comment generated here */
}
```

3.3.16 8-bit interval timer

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

Table 3.17 API Functions: [8-Bit Interval Timer]

API Function Name	Function
R_IT8Bitm_Channeln_Create	Performs initialization necessary to control the 8-bit interval timer.
R_IT8Bitm_Channeln_Create_UserInit	Performs user-defined initialization relating to the 8-bit interval timer.
r_it8bitm_channeln_interrupt	Performs processing in response to the 8-bit interval timer interrupt INTIT*n*0 or INTIT*n*1.
R_IT8Bitm_Channeln_Start	Starts the count of the 8-bit interval timer.
R_IT8Bitm_Channeln_Stop	Ends the count of the 8-bit interval timer.
R_IT8Bitm_Channeln_Set_PowerOff	Halts the clock supplied to the 8-bit interval timer.
R_IT8Bitm_Set_PowerOff	Halts the clock supplied to the 8-bit interval timer.

R_IT8Bit*m*_Channel*n*_Create

Performs initialization necessary to control the 8-bit interval timer.

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

[Syntax]

```
void        R_IT8Bitm_Channeln_Create ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IT8Bit*m*_Channel*n*_Create_UserInit

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

Remark This API function is called as the R_IT8bitm_Channeln_Create callback routine.

[Syntax]

void	R_IT8Bit*m*_Channel*n*_Create_UserInit (void);

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_it8bit*m*_channel*n*_interrupt

Performs processing in response to the 8-bit interval timer interrupt INTIT*n*0 or INTIT*n*1.

Remark This API function is called as the interrupt process corresponding to the 8-bit interval timer interrupt INTIT*n*0 or INTIT*n*1.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_it8bitm_channeln_interrupt ( void );
```

CC-RL Compiler

```
static void    __near r_it8bitm_channeln_interrupt ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IT8Bit*m*_Channel*n*_Start

Starts the count of the 8-bit interval timer.

[Syntax]

```
void        R_IT8Bitm_Channeln_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IT8Bit*m*_Channel*n*_Stop

Ends the count of the 8-bit interval timer.

[Syntax]

```
void        R_IT8Bitm_Channeln_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IT8Bit*m*_Channel*n*_Set_PowerOff

Halts the clock supplied to the 8-bit interval timer.

Remark Calling this API function changes the 8-bit interval timer to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void        R_IT8Bitm_Channeln_Set_PowerOff ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IT8Bit*m*_Set_PowerOff

Halts the clock supplied to the 8-bit interval timer.

Remark Calling this API function changes the 8-bit interval timer to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void    R_IT8Bitm_Set_PowerOff ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

Usage example

Use timer as One-shot timer.

[GUI setting example]

8bit interval timer				Used
	IT8bit0			Used
		Channel 0		Used
			Channel 0	8 bit
			Interval timer operation setting	Auto
			Interval value	100ms (Actual value：7.8125)
			Detection of compare match (INTIT00)	Used
			Priority	Low

[API setting example]

r_cg_main.c

```
void main(void)
{
        R_MAIN_UserInit();
        /* Start user code. Do not edit comment generated here */
        /* Start 8 bit interval timer unit0 Channel0 operation */
        R_IT8Bit0_Channel0_Start();

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

r_cg_it8bit_user.c

```
static void __near r_it8bit0_channel0_interrupt(void)
{
        /* Start user code. Do not edit comment generated here */
        /* Stop 8 bit interval timer unit0 Channel0 operation */
        R_IT8Bit0_Channel0_Stop();
        /* End user code. Do not edit comment generated here */
}
```

3.3.17 16-bit wakeup timer

Below is a list of API functions output by the Code Generator for 16-bit wakeup timer (WUTM) use.

Table 3.18 API Functions: [16-bit Wakeup Timer]

API Function Name	Function
R_WUTM_Create	Performs initialization necessary to control the 16-bit wakeup timer.
R_WUTM_Create_UserInit	Performs user-defined initialization relating to the 16-bit wakeup timer.
r_wutm_interrupt	Performs processing in response to the timer interrupt.
R_WUTM_Start	Starts the count for 16-bit wakeup timer.
R_WUTM_Stop	Ends the count for 16-bit wakeup timer.
R_WUTM_Set_PowerOff	Halts the clock supplied to the 16-bit wakeup timer.

R_WUTM_Create

Performs initialization necessary to control the 16-bit wakeup timer.

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

[Syntax]

```
void R_WUTM_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_WUTM_Create_UserInit

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

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

[Syntax]

```
void R_WUTM_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_wutm_interrupt

Performs processing in response to the timer interrupt.

Remark This API function is called as the interrupt process corresponding to the timer interrupt.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_wutm_interrupt ( void );
```

CC-RL Compiler

```
static void     __near r_wutm_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_WUTM_Start

Starts the count for 16-bit wakeup timer.

[Syntax]

```
void            R_WUTM_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_WUTM_Stop

Ends the count for 16-bit wakeup timer.

[Syntax]

```
void        R_WUTM_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_WUTM_Set_PowerOff

Halts the clock supplied to the 16-bit wakeup timer.

Remark Calling this API function changes the 16-bit wakeup timer to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void        R_WUTM_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

Usage example

Use timer as One-shot timer.

[GUI setting example]

Timer				Used
	TAU0			Unused
	WUTM			Used
			16-bit wake-up timer operation setting	Used
			Count clock setting	Auto
			Interval value	100ms (Actual value：100)
			WUTM and WUTMCMP match, generate an interrupt (INTWUTM)	Used
			Priority(INTWUTM)	Low

[API setting example]

r_main.c

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

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

r_cg_timer_user.c

```
static void __near r_wutm_interrupt(void)
{
        /* Start user code. Do not edit comment generated here */
        /* Stop WUTM counter */
        R_WUTM_Stop();
        /* End user code. Do not edit comment generated here */
}
```

3.3.18 Clock output/buzzer output controller

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

Table 3.19 API Functions: [Clock Output/Buzzer Output Controller]

API Function Name	Function
R_PCLBUZn_Create	Performs initialization necessary to control the clock/buzzer output controller.
R_PCLBUZn_Create_UserInit	Performs user-defined initialization relating to the clock/buzzer output controller.
R_PCLBUZn_Start	Starts clock/buzzer output.
R_PCLBUZn_Stop	Ends clock/buzzer output.
R_PCLBUZn_Set_PowerOff	Halts the clock supplied to the clock/buzzer output controller.

R_PCLBUZ*n*_Create

Performs initialization necessary to control the clock/buzzer output controller.

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

[Syntax]

```
void        R_PCLBUZn_Create ( void );
```

Remark *n* is the output pin.

[Argument(s)]

None.

[Return value]

None.

R_PCLBUZ*n*_Create_UserInit

Performs user-defined initialization relating to the clock/buzzer output controller.

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

[Syntax]

```
void        R_PCLBUZn_Create_UserInit ( void );
```

Remark *n* is the output pin.

[Argument(s)]

None.

[Return value]

None.

R_PCLBUZ*n*_Start

Starts clock/buzzer output.

[Syntax]

```
void        R_PCLBUZn_Start ( void );
```

Remark *n* is the output pin.

[Argument(s)]

None.

[Return value]

None.

R_PCLBUZ*n*_Stop

Ends clock/buzzer output.

[Syntax]

```
void        R_PCLBUZn_Stop ( void );
```

Remark *n* is the output pin.

[Argument(s)]

None.

[Return value]

None.

R_PCLBUZ*n*_Set_PowerOff

Halts the clock supplied to the clock/buzzer output controller.

Remark 1. Calling this API function changes the clock/buzzer output controller to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

Remark 2. This API function stops the clock supply to the clock/buzzer output controller, by operating the RTCEN bit of peripheral enable register *n*.
For this reason, this API function also stops the clock supply to other peripheral devices sharing the RTCEN bit (e.g. real-time clock).

[Syntax]

```
void        R_PCLBUZn_Set_PowerOff ( void );
```

Remark *n* is the output pin.

[Argument(s)]

None.

[Return value]

None.

Usage example

Start clock and buzzer output

[GUI setting example]

Clock output/Buzzer output				Used
	PCLBUZ0			Used
			Clock output/buzzeroutput operation setting	Used
			PCLBUZ0 output clock selection	1.875 (fSL/2^3)(kHz)
			Slow mode	Unused

[API setting example]

r_main.c

```
void main(void)
{
        R_MAIN_UserInit();
        /* Start user code. Do not edit comment generated here */
        /* Start the PCLBUZ0 module */
        R_PCLBUZ0_Start();

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

3.3.19 Watchdog timer

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

Table 3.20 API Functions: [Watchdog Timer]

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

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_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_interrupt

Performs processing in response to the interval interrupt INTWDTI.

Remark This API function is called as the interrupt process corresponding to the interval interrupt INTWDTI.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_wdt_interrupt ( void );
```

CC-RL Compiler

```
static void    __near r_wdt_interrupt ( 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.

Usage example

Clear the flag in the interval interrupt of Watchdog timer when the flag is ‘1’.

[GUI setting example]

Watchdog timer				Used
	WDT			Used
			Watchdog timer operation setting	Used
			Operation in HALT/STOP/SNOOZE mode setting	Enabled
			Overflow time	136.53 (2^11/fWDT)(ms)
			Window open period	100(%)
			Enable interval interrupt when 75% + 1/2fIL of overflow time (INTWDTI)	Used
			Priority	Low

[API setting example]

r_main.c

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

r_cg_wdt_user.c

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

static void __near r_wdt_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    if(g_wdt_f == 1U)
    {
        /* Restart the watchdog timer */
        R_WDT_Restart();
    }
    /* End user code. Do not edit comment generated here */
}
```

3.3.20 24-bit DS A/D converter with programmable gain instrumentation amplifier

Below is a list of API functions output by the Code Generator for 24-bit ΔΣ A/D converter with programmable gain instrumentation amplifier use.

Table 3.21 API Functions: [24-bit ΔΣ A/D converter with programmable gain instrumentation amplifier]

API Function Name	Function
R_PGA_DSAD_Create	Performs initialization necessary to control the 24-bit ΔΣ A/D converter with programmable gain instrumentation amplifier.
R_PGA_DSAD_Create_UserInit	Performs user-defined initialization relating to the 24-bit ΔΣ A/D converter with programmable gain instrumentation amplifier.
r_pga_dsad_interrupt_conversion	Performs processing in response to the 24-bit ΔΣ A/D conversion end interrupt INTDSAD.
r_pga_dsad_interrupt_scan	Performs processing in response to the 24-bit ΔΣ A/D scan end interrupt INTDSADS.
R_PGA_DSAD_Start	Starts A/D conversion.
R_PGA_DSAD_Stop	Ends A/D conversion.
R_PGA_DSAD_Set_PowerOff	Halts the clock supplied to the 24-bit ΔΣ A/D converter with programmable gain instrumentation amplifier.
R_PGA_DSAD_Get_AverageResult	Reads the results of A/D conversion.(mean value)
R_PGA_DSAD_Get_Result	Reads the results of A/D conversion.
R_PGA_DSAD_CAMP_OffsetTrimming	Connects the configurable amplifier to the 24-bit ΔΣ A/D converter and trims the offset.
r_pga_dsad_conversion_interrupt	Performs processing in response to the 24-bit ΔΣ A/D conversion end interrupt INTDSAD.
r_pga_dsad_scan_interrupt	Performs processing in response to the 24-bit ΔΣ A/D scan end interrupt INTDSADS.

R_PGA_DSAD_Create

Performs initialization necessary to control the 24-bit ΔΣ A/D converter with programmable gain instrumentation amplifier.

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

[Syntax]

```
void        R_PGA_DSAD_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA_DSAD_Create_UserInit

Performs user-defined initialization relating to the 24-bit ΔΣ A/D converter with programmable gain instrumentation amplifier.

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

[Syntax]

```
void        R_PGA_DSAD_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_pga_dsad_interrupt_conversion

Performs processing in response to the 24-bit ΔΣ A/D conversion end interrupt INTDSAD.

Remark This API function is called as the interrupt process corresponding to the 24-bit ΔΣ A/D conversion end interrupt INTDSAD.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_pga_dsad_interrupt_conversion ( void );
```

CC-RL Compiler

```
static void    __near r_pga_dsad_interrupt_conversion ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_pga_dsad_interrupt_scan

Performs processing in response to the 24-bit ΔΣ A/D scan end interrupt INTDSADS.

Remark This API function is called as the interrupt process corresponding to the 24-bit ΔΣ A/D scan end interrupt INTDSADS.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_pga_dsad_interrupt_scan ( void );
```

CC-RL Compiler

```
static void     __near r_pga_dsad_interrupt_scan ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA_DSAD_Start

Starts A/D conversion.

[Syntax]

```
void        R_PGA_DSAD_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA_DSAD_Stop

Ends A/D conversion.

[Syntax]

```
void          R_PGA_DSAD_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA_DSAD_Set_PowerOff

Halts the clock supplied to the 24-bit ΔΣ A/D converter with programmable gain instrumentation amplifier.

[Syntax]

```
void        R_PGA_DSAD_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA_DSAD_Get_AverageResult

Reads the average results of A/D conversion.(mean value)

[Syntax]

```
#include    "r_cg_macrodriver.h"
void        R_PGA_DSAD_Get_AverageResult ( uint16_t * const bufferH, uint16_t * const bufferL );
```

[Argument(s)]

I/O	Argument	Description
O	uint16_t * const *bufferH*;	Pointer to area in which to store read results of A/D conversion (DSADMVM resister and DSADMVH resister)
O	uint16_t * const *bufferLl*;	Pointer to area in which to store read results of A/D conversion (DSADMVC resister and DSADMVL resister)

[Return value]

None.

R_PGA_DSAD_Get_Result

Reads the results of A/D conversion.

[Syntax]

```
#include   "r_cg_macrodriver.h"
void       R_PGA_DSAD_Get_Result ( uint16_t * const bufferH, uint16_t * const bufferL );
```

[Argument(s)]

I/O	Argument	Description
O	uint16_t * const *bufferH*;	Pointer to area in which to store the results of A/D conversion (DSADCRM resister and DSADCRH resister)
O	uint16_t * const *bufferLl*;	Pointer to area in which to store the results of A/D conversion (DSADCRC resister and DSADCRL resister)

[Return value]

None.

R_PGA_DSAD_CAMP_OffsetTrimming

Connects the configurable amplifier to the 24-bit ΔΣ A/D converter and trims the offset.

[Syntax]

```
void        R_PGA_DSAD_CAMP_OffsetTrimming ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_pga_dsad_conversion_interrupt

Performs processing in response to the 24-bit ΔΣ A/D conversion end interrupt INTDSAD.

Remark This API function is called as the interrupt process corresponding to the 24-bit ΔΣ A/D conversion end interrupt INTDSAD.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_pga_dsad_conversion_interrupt ( void );
```

CC-RL Compiler

```
static void    __near r_pga_dsad_conversion_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_pga_dsad_scan_interrupt

Performs processing in response to the 24-bit ΔΣ A/D scan end interrupt INTDSADS.

Remark This API function is called as the interrupt process corresponding to the 24-bit ΔΣ A/D scan end interrupt INTDSADS.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_pga_dsad_scan_interrupt ( void );
```

CC-RL Compiler

```
static void    __near r_pga_dsad_scan_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.3.21 A/D converter

Below is a list of API functions output by the Code Generator for A/D converter use.

Table 3.22 API Functions: [A/D Converter]

API Function Name	Function
R_ADC_Create	Performs initialization necessary to control the A/D converter.
R_ADC_Create_UserInit	Performs user-defined initialization relating to the A/D converter.
r_adc_interrupt	Performs processing in response to the A/D conversion end interrupt INTAD.
R_ADC_Set_OperationOn	Enables operation of voltage converter.
R_ADC_Set_OperationOff	Disables operation of voltage converter.
R_ADC_Start	Starts A/D conversion.
R_ADC_Stop	Ends A/D conversion.
R_ADC_Reset	Reset A/D conversion.
R_ADC_Set_PowerOff	Halts the clock supplied to the A/D converter.
R_ADC_Set_ADChannel	Configures the analog voltage input pin for A/D conversion.
R_ADC_Set_SnoozeOn	Enables the switch from STOP mode to SNOOZE mode.
R_ADC_Set_SnoozeOff	Disables the switch from STOP mode to SNOOZE mode.
R_ADC_Set_TestChannel	Sets the operation mode of A/D converter.
R_ADC_Get_Result	Reads the results of A/D conversion (10 bits).
R_ADC_Get_Result_8bit	Reads the results of A/D conversion (10 bits).

R_ADC_Create

Performs initialization necessary to control the A/D converter.

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

[Syntax]

```
void        R_ADC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Create_UserInit

Performs user-defined initialization relating to the A/D converter.

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

[Syntax]

```
void        R_ADC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_adc_interrupt

Performs processing in response to the A/D conversion end interrupt INTAD.

Remark This API function is called as the interrupt process corresponding to the A/D conversion end interrupt INTAD.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_adc_interrupt ( void );
```

CC-RL Compiler

```
static void    __near r_adc_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Set_OperationOn

Enables operation of voltage converter.

Remark 1. About 1 microsecond of stabilization time is required when changing the voltage converter from operation stopped to operation enabled status.
Consequently, about 1 micro second must be left free between the call to this API function and the call to R_ADC_Start.

Remark 2. On the [A/D Converter], in the [Comparator operation setting] area, if "Operation" is selected, then the voltage converter will be switched to "always on".
There is thus no need to call this API function in this case.

[Syntax]

```
void        R_ADC_Set_OperationOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Set_OperationOff

Disables operation of voltage converter.

[Syntax]

```
void        R_ADC_Set_OperationOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Start

Starts A/D conversion.

Remark About 1 micro second of stabilization time is required when changing the voltage converter from operation stopped to operation enabled status.
Consequently, about 1 micro second must be left free between the call to R_ADC_Set_OperationOn and the call to this API function.

[Syntax]

```
void        R_ADC_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Stop

Ends A/D conversion.

Remark The voltage converter continues to operate after the process of this API function completes.
Consequently, to stop the operation of the voltage converter, you must call R_ADC_Set_OperationOff after the process of this API function completes.

[Syntax]

```
void        R_ADC_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Reset

Reset A/D conversion.

[Syntax]

```
void        R_ADC_Reset ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Set_PowerOff

Halts the clock supplied to the A/D converter.

Remark Calling this API function changes the A/D converter to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void        R_ADC_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Set_ADChannel

Configures the analog voltage input pin for A/D conversion.

Remark The value specified in argument channel is set to analog input *channel* specification register (ADS).

[Syntax]

```
#include        "r_cg_macrodriver.h"
#include        "r_cg_adc.h"
MD_STATUS   R_ADC_Set_ADChannel ( ad_channel_t channel );
```

[Argument(s)]

I/O	Argument	Description
I	ad_channel_t *channel*;	Analog voltage input pin ADCHANNEL*n* : Input pin

Remark See the header file r_cg_adc.h for details about the analog voltage input pin ADCHANNEL*n*.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_ADC_Set_SnoozeOn

Enables the switch from STOP mode to SNOOZE mode.

[Syntax]

```
void        R_ADC_Set_SnoozeOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Set_SnoozeOff

Disables the switch from STOP mode to SNOOZE mode.

[Syntax]

```
void        R_ADC_Set_SnoozeOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_ADC_Set_TestChannel

Sets the operation mode of A/D converter.

[Syntax]

```
#include        "r_cg_macrodriver.h"
#include        "r_cg_adc.h"
MD_STATUS   R_ADC_Set_TestChannel ( test_channel_t channel );
```

[Argument(s)]

I/O	Argument	Description
I	test_channel_t *channel*;	Operation mode of A/D converter ADNORMALINPUT : Normal mode (Normal A/D conversion) ADNORMALINPUT : Test mode (AVREFM input) ADNORMALINPUT : Test mode (AVREFP input)

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_ADC_Get_Result

Reads the results of A/D conversion (10 bits).

[Syntax]

```
#include    "r_cg_macrodriver.h"
void        R_ADC_Get_Result ( uint16_t * const buffer );
```

[Argument(s)]

I/O	Argument	Description
O	uint16_t * const *buffer*;	Pointer to area in which to store read results of A/D conversion

[Return value]

None.

R_ADC_Get_Result_8bit

Reads the results of A/D conversion (8 bits; most significant 8 bits of 10-bit resolution).

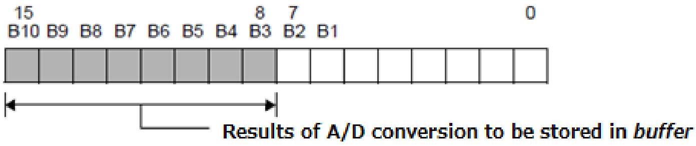
[Syntax]

```
#include    "r_cg_macrodriver.h"
void        R_ADC_Get_Result ( uint8_t * const buffer);
```

[Argument(s)]

I/O	Argument	Description
O	Uint8_t * const *buffer*;	Pointer to area in which to store the results of A/D conversion

Remark Below is an example of the results of A/D conversion to be stored in *buffer*.



[Return value]

None.

Usage example

Get the A/D conversion result of 2 pins alternately.

[GUI setting example]

A/D convertor				Used
	ADC			Used
			A/D convertor operation setting	Used
			Comparator operation setting	Operation
			Resolution	8 bits
			VREF(+) setting	VDD
			VREF(-) setting	VSS
			Trigger setting mode	Software trigger mode
			Operation mode setting	One-shot select mode
			ANI0 - ANI23 analog input selection	ANI0 - ANI1
			A/D channel selection	ANI0
			Conversion time mode	Normal 1
			Conversion time	34 (1088/fCLK)(μs)
			Conversion result upper/lower bound value setting	Generates an interrupt request (INTAD) when ADLL≦ADCRH≦ADUL
			Upper bound (ADUL) value	255
			Lower bound (ADLL) value	0
			Use A/D interrupt (INTAD)	Used
			Priority	Low

[API setting example]

r_main.c

```
void main(void)
{
        R_MAIN_UserInit();
        /* Start user code. Do not edit comment generated here */
        /* Start the AD converter */
        R_ADC_Start();

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

r_cg_adc_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_adc_ch000_value;
volatile uint8_t g_adc_ch001_value;
/* End user code. Do not edit comment generated here */

static void __near r_adc_interrupt(void)
{
        /* Start user code. Do not edit comment generated here */
        /* Stop the AD converter */
        R_ADC_Stop();

        if(ADS == (uint8_t)ADCHANNEL0)
        {
                /* Return the higher 8 bits conversion result */
                R_ADC_Get_Result_8bit((uint8_t *)&g_adc_ch000_value);

                /* Start the AD converter */
                R_ADC_Set_ADChannel(ADCHANNEL1);
                R_ADC_Start();
        }
        else
        {
                /* Return the higher 8 bits conversion result */
                R_ADC_Get_Result_8bit((uint8_t *)&g_adc_ch001_value);
        }
        /* End user code. Do not edit comment generated here */
}
```

3.3.22 Configurable amplifier

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

Table 3.23 API Functions: [Configurable amplifier]

API Function Name	Function
R_CAMP_Create	Performs initialization necessary to control the configurable amplifier.
R_CAMP_Create_UserInit	Performs user-defined initialization relating to the configurable amplifier.
R_CAMPn_Start	Turns on the power of the configurable amplifier(AMP*n*).
R_CAMPn_Stop	Turns off the power of the configurable amplifier(AMP*n*).
R_CAMP_Set_PowerOff	Halts the clock supplied to the configurable amplifier.

R_CAMP_Create

Performs initialization necessary to control the configurable amplifier.

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

[Syntax]

```
void        R_CAMP_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CAMP_Create_UserInit

Performs user-defined initialization relating to the configurable amplifier.

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

[Syntax]

```
void        R_CAMP_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_CAMPn_Start

Turns on the power of the configurable amplifier(AMP*n*).

[Syntax]

```
void        R_CAMPn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CAMP*n*_Stop

Turns off the power of the configurable amplifier(AMP*n*).

[Syntax]

```
void        R_CAMPn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CAMP_Set_PowerOff

Halts the clock supplied to the configurable amplifier.

[Syntax]

```
void        R_CAMP_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.3.23 Temperature sensor

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

Table 3.24 API Functions: [Temperature Sensor]

API Function Name	Function
R_TMPS_Create	Performs initialization necessary to control the temperature sensor.
R_TMPS_Create_UserInit	Performs user-defined initialization relating to the temperature sensor.
R_TMPS_Start	Starts measurement of the temperature that uses the temperature sen sor.
R_TMPS_Stop	Ends measurement of the temperature that uses the temperature sensor.
R_TMPS_Reset	Reset the temperature sensor.
R_TMPS_Set_PowerOff	Halts the clock supplied to the temperature sensor.

R_TMPS_Create

Performs initialization necessary to control the temperature sensor.

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

[Syntax]

```
void        R_TMPS_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMPS_Create_UserInit

Performs user-defined initialization relating to the temperature sensor.

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

[Syntax]

```
void          R_TMPS_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMPS_Start

Starts measurement of the temperature that uses the temperature sensor.

[Syntax]

```
void        R_TMPS_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMPS_Stop

Ends measurement of the temperature that uses the temperature sensor.

[Syntax]

```
void        R_TMPS_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMPS_Reset

Reset the temperature sensor.

[Syntax]

```
void          R_TMPS_Reset ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_TMPS_Set_PowerOff

Halts the clock supplied to the temperature sensor.

Remark Calling this API function changes the temperature sensor to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void        R_TMPS_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

Usage example

Measure temperature by measuring the output voltage from the temperature sensor using the A/D converter.

[GUI setting example]

Temparature sensor				Used
	TS			Used
			Temparature sensor operation setting	Used
			Operation mode setting	Normal-temparature range (mode 2)

A/Dconvertor				Used
	ADC			Used
			A/D convertor operation setting	Used
			Comparator operation setting	Stop
			Resolution setting	8 bits
			VREF(+) setting	Internal reference voltage
			VREF(-) setting	VSS
			Trigger mode setting	Software trigger mode
			Operation mode setting	On-shot select mode
			ANI0 - ANI5Analog input selection	All digital
			A/D channel selection	Temparature sensor output voltage
			Conversion time mode	Normal 1
			Conversion time	544/fCLK 22.6667(μs)
			Conversion result upper/lower bound value setting	Generates an interrupt request (INTAD) when ADLL≦ADCRH≦ADUL
			Upper bound (ADUL) value	255
			Lower bound (ADLL) value	0
			Use A/D interrupt (INTAD)	Used
			Priority	Low

[API setting example]

r_cg_main.c

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

	/* Start the AD converter */
	R_ADC_Start();

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

r_cg_adc_user.c

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

static void __near r_adc_interrupt(void)
{
	/* Start user code. Do not edit comment generated here */
	/* Stop the AD converter */
	R_ADC_Stop();

	/* Return the higher 8 bits conversion result */
	R_ADC_Get_Result_8bit((uint8_t *)&g_adc_value);
	/* End user code. Do not edit comment generated here */
}
```

3.3.24 24-bit DS A/D converter

Below is a list of API functions output by the Code Generator for 24-bit ΔΣ A/D converter use.

Table 3.25 API Functions: [224-bit ΔΣ A/D Converter]

API Function Name	Function
R_DSADC_Create	Performs initialization necessary to control the 24-bit ΔΣ A/D converter.
R_DSADC_Create_UserInit	Performs user-defined initialization relating to the 24-bit ΔΣ A/D converter.
r_dsadc_interrupt	Performs processing in response to the ΔΣ A/D conversion end interrupt INTDSAD.
r_dsadzcn_interrupt	Performs processing in response to the zero-cross detection interrupt INTDSADZC*n*.
R_DSADC_Set_OperationOn	Enables operation of 24-bit ΔΣ A/D converter.
R_DSADC_Set_OperationOff	Disables operation of 24-bit ΔΣ A/D converter.
R_DSADC_Start	Starts A/D conversion.
R_DSADC_Stop	Ends A/D conversion.
R_DSADC_Reset	Reset the 24-bit ΔΣ A/D converter.
R_DSADC_Set_PowerOff	Performs electric charge reset for the 24-bit ΔΣ A/D converter.
R_DSADC_Channeln_Get_Result	Reads the results of A/D conversion (24 bits).
R_DSADC_Channeln_Get_Result_16bit	Reads the results of A/D conversion (16 bits; most significant 16 bits of 24-bit resolution).

R_DSADC_Create

Performs initialization necessary to control the 24-bit ΔΣ A/D converter.

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

[Syntax]

```
void        R_DSADC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DSADC_Create_UserInit

Performs user-defined initialization relating to the 24-bit ΔΣ A/D converter.

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

[Syntax]

```
void        R_DSADC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_dsadc_interrupt

Performs processing in response to the ΔΣ A/D conversion end interrupt INTDSAD.

Remark This API function is called as the interrupt process corresponding to the ΔΣ A/D conversion end interrupt INTDSAD.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_dsadc_interrupt ( void );
```

CC-RL Compiler

```
static void     __near r_dsadc_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_dsadzcn_interrupt

Performs processing in response to the zero-cross detection interrupt INTDSADZC*n*.

Remark This API function is called as the interrupt process corresponding to the zero-cross detection interrupt INTDSADZC*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_dsadzcn_interrupt ( void );
```

CC-RL Compiler

```
static void     __near r_dsadzcn_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DSADC_Set_OperationOn

Enables operation of 24-bit ΔΣ A/D converter.

[Syntax]

```
void        R_DSADC_Set_OperationOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DSADC_Set_OperationOff

Disables operation of 24-bit ΔΣ A/D converter.

[Syntax]

```
void        R_DSADC_Set_OperationOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DSADC_Start

Starts A/D conversion.

[Syntax]

```
void        R_DSADC_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DSADC_Stop

Ends A/D conversion.

[Syntax]

```
void        R_DSADC_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DSADC_Reset

Reset the 24-bit ΔΣ A/D converter.

[Syntax]

```
void        R_DSADC_Set_Reset ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DSADC_Set_PowerOff

Performs electric charge reset for the 24-bit ΔΣ A/D converter.

Remark About 1.4 microseconds of stabilization time is required when electric charge reset is performed for the 24-bit ΔΣ A/D converter.

[Syntax]

```
void        R_DSADC_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DSADC_Channel*n*_Get_Result

Reads the results of A/D conversion (24 bits).

Remark The result of A/D conversion (24 bits) by this API function must be read within the maximum pending time of the ΔΣ A/D conversion result register n after ΔΣ A/D conversion end interrupt INTDSAD is generated.

[Syntax]

```
#include   "r_cg_macrodriver.h"
void       R_DSADC_Channeln_Get_Result ( uint32_t * const buffer );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	uint32_t * const *buffer*;	Pointer to area in which to store read results of A/D conversion

[Return value]

None.

R_DSADC_Channel*n*_Get_Result_16bit

Reads the results of A/D conversion (16 bits; most significant 16 bits of 24-bit resolution).

Remark The result of A/D conversion by this API function must be read within the maximum pending time of the ΔΣ A/D conversion result register n after ΔΣ A/D conversion end interrupt INTDSAD is generated.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void        R_DSADC_Channeln_Get_Result_16bit ( uint16_t * const buffer );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	Uint16_t * const *buffer*;	Pointer to area in which to store the results of A/D conversion

[Return value]

None.

3.3.25 D/A converter

Below is a list of API functions output by the Code Generator for D/A converter use.

Table 3.26 API Functions: [D/A Converter]

API Function Name	Function
R_DAC_Create	Performs initialization necessary to control the D/A converter.
R_DAC_Create_UserInit	Performs user-defined initialization relating to the D/A converter.
R_DACn_Start	Starts D/A conversion.
R_DACn_Stop	Ends D/A conversion.
R_DAC_Set_PowerOff	Halts the clock supplied to the D/A converter.
R_DACn_Set_ConversionValue	Sets the analog voltage output to the ANO*n* pin.
R_DACn_Change_OutputVoltage_8bit	Changes the output voltage of D/A converter.(8bit mode)
R_DACn_Change_OutputVoltage	Changes the output voltage of D/A converter.(12bit mode)
R_DACn_Create	Performs initialization necessary to control the D/A converter.
R_DAC_Reset	Reset the D/A converter.
R_DACn_Create_UserInit	Performs user-defined initialization relating to the D/A converter.

R_DAC_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_DAC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DAC_Create_UserInit

Performs user-defined initialization relating to the D/A converter.

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

[Syntax]

```
void R_DAC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DAC*n*_Start

Starts D/A conversion.

[Syntax]

```
void        R_DACn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DAC*n*_Stop

Ends D/A conversion.

[Syntax]

```
void        R_DACn_Stop ( void );
```

Remark   *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DAC_Set_PowerOff

Halts the clock supplied to the D/A converter.

Remark Calling this API function changes the D/A converter to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void        R_DAC_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DAC*n*_Set_ConversionValue

Sets the analog voltage output to the ANO*n* pin.

[Syntax]

```
#include   "r_cg_macrodriver.h"
void       R_DACn_Set_ConversionValue ( uint8_t reg_value );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	Uint8_t *reg_value*;	D/A conversion value (0x0 to 0xFF)

[Return value]

None.

R_DAC*n*_Change_OutputVoltage_8bit

Changes the output voltage of D/A converter.(8bit mode)

[Syntax]

```
#include    "r_cg_macrodriver.h"
void        R_DACn_Change_OutputVoltage_8bit ( uint8_t outputVoltage );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint8_t *outputVoltage*;	output Voltage (Low 8bit)

[Return value]

None.

R_DAC*n*_Change_OutputVoltage

Changes the output voltage of D/A converter.(12bit mode)

[Syntax]

```
#include   "r_cg_macrodriver.h"
void       R_DACn_Change_OutputVoltage ( uint16_t outputVoltage );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint16_t *outputVoltage*;	output Voltage (Low 12bit)

[Return value]

None.

R_DAC*n*_Create

Performs initialization necessary to control the D/A converter.

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

[Syntax]

```
#include    "r_cg_macrodriver.h"
void        R_DACn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DAC_Reset

Reset the D/A converter.

[Syntax]

```
void            R_DAC_Reset ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DAC*n*_Create_UserInit

Performs user-defined initialization relating to the D/A converter.

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

[Syntax]

```
void        R_DACn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

Usage example

Start conversion digital input to analog signal from 0x00. Add 0x10 to digital input at fixed interval. Stop conversion when digital input becomes 0xFF.

[GUI setting example]

D/A convertor				Used
	DAC			Used
		DAC0		
			D/A convertor operation setting	Used
			D/A convertor operation setting	Normal mode
			Analog output (ANO0)	Used
			Conversion value	0x00

Timer				Used
	TAU0			Used
		Channel 0		
			channel 0	Interval timer
			Interval value (16 bits)	100ms (Actual value : 100)
			Generates INTTM00 when counting is started	Unused
			End of timer channel 0 count, generate an interrupt (INTTM00)	Used
			Priority (INTTM00)	Low

[API setting example]

r_main.c

```
void main(void)
{
        R_MAIN_UserInit();
        /* Start user code. Do not edit comment generated here */
        /* Start TAU0 channel 0 counter */
        R_TAU0_Channel0_Start();

        /* Enable the DA converter channel 0 */
        R_DAC0_Start();

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

r_cg_timer_user.c

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

/* Start user code for global. Do not edit comment generated here */
volatile uint16_t g_dac0_value = _00_DA0_CONVERSION_VALUE;
/* End user code. Do not edit comment generated here */

static void __near r_tau0_channel0_interrupt(void)
{
        /* Start user code. Do not edit comment generated here */
        g_dac0_value += 0x0010U;
        if (g_dac0_value <= 0x00FFU)
        {
                /* Set the DA converter channel 0 value */
                R_DAC0_Set_ConversionValue((uint8_t)g_dac0_value);
        }
        else
        {
                /* Stop the DA converter channel 0 */
                R_DAC0_Stop();

                /* Stop TAU0 channel 0 counter */
                R_TAU0_Channel0_Stop();
        }
        /* End user code. Do not edit comment generated here */
}
```

3.3.26 Programmable gain amplifier

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

Table 3.27 API Functions: [Programmable Gain Amplifier]

API Function Name	Function
R_PGA_Create	Performs initialization necessary to control the programmable gain amplifier.
R_PGA_Create_UserInit	Performs user-defined initialization relating to the programmable gain amplifier.
R_PGA_Start	Starts the operation of programmable gain amplifier.
R_PGA_Stop	Ends the operation of programmable gain amplifier.
R_PGA_Reset	Reset the programmable gain amplifier.
R_PGA_Set_PowerOff	Halts the clock supplied to the programmable gain amplifier.

R_PGA_Create

Performs initialization necessary to control the programmable gain amplifier.

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

[Syntax]

```
void        R_PGA_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA_Create_UserInit

Performs user-defined initialization relating to the programmable gain amplifier.

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

[Syntax]

```
void R_PGA_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA_Start

Starts the operation of programmable gain amplifier.

[Syntax]

```
void        R_PGA_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA_Stop

Ends the operation of programmable gain amplifier.

[Syntax]

```
void        R_PGA_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA_Reset

Reset the operation of programmable gain amplifier.

[Syntax]

```
void        R_PGA_Reset ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA_Set_PowerOff

Halts the clock supplied to the programmable gain amplifier.

Remark Calling this API function changes the comparator to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void        R_PGA_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.3.27 Comparator

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

Table 3.28 API Functions: [Comparator]

API Function Name	Function
R_COMP_Create	Performs initialization necessary to control the comparator.
R_COMP_Create_UserInit	Performs user-defined initialization relating to the comparator.
r_compn_interrupt	Performs processing in response to the comparator interrupt INTCMP*n*.
R_COMPn_Start	Begins comparison of reference input voltage and analog input voltage.
R_COMPn_Stop	Stops comparison of reference input voltage and analog input voltage.
R_COMP_Reset	Reset the comparator.
R_COMP_Set_PowerOff	Halts the clock supplied to the comparator.

R_COMP_Create

Performs initialization necessary to control the comparator.

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

[Syntax]

```
void R_COMP_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_COMP_Create_UserInit

Performs user-defined initialization relating to the comparator.

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

[Syntax]

```
void        R_COMP_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_comp*n*_interrupt

Performs processing in response to the comparator interrupt INTCMP*n*.

Remark This API function is called as the interrupt process corresponding to the comparator interrupt INTCMP*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_compn_interrupt ( void );
```

CC-RL Compiler

```
static void    __near r_compn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_COMP*n*_Start

Begins comparison of reference input voltage and analog input voltage.

[Syntax]

```
void        R_COMPn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_COMP*n*_Stop

Stops comparison of reference input voltage and analog input voltage.

[Syntax]

```
void        R_COMPn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_COMP_Reset

Reset the comparator.

[Syntax]

```
void        R_COMP_Reset ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_COMP_Set_PowerOff

Halts the clock supplied to the comparator.

Remark Calling this API function changes the comparator to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void        R_COMP_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

Usage example

Stop the comparator after the valid edge of comparison result.

[GUI setting example]

Comparator				Used
	Comparator			Used
			Comparator input	IVCMP00
			Reference voltage	IVREF0
			Enable digital filter	Unused
			Output setting (VCOUT0)	Used
			Internal output polarity setting	Normal
			STOP mode release setting	Used
			When detecting the valid edge of the comparator output, generate an interrupt (INTCMP0)	Used
			Valid edge detection	Both edges
			Priority	Low

[API setting example]

r_main.c

```
void main(void)
{
        R_MAIN_UserInit();
        /* Start user code. Do not edit comment generated here */
        /* Start the comparator 0 */
        R_COMP0_Start();

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

r_cg_comp_user.c

```
static void __near r_comp0_interrupt(void)
{
        /* Start user code. Do not edit comment generated here */
        /* Stop the comparator 0 */
        R_COMP0_Stop();
        /* End user code. Do not edit comment generated here */
}
```

3.3.28 Comparator/ProgrammableGainAmplifier

Below is a list of API functions output by the Code Generator for comparator/programmable gain amplifier use.

Table 3.29 API Functions: [Comparator/ProgrammableGainAmplifier]

API Function Name	Function
R_COMPPGA_Create	Performs initialization necessary to control the comparator/programmable gain amplifier.
R_COMPPGA_Set_PowerOff	Halts the clock supplied to the comparator/programmable gain amplifier.
R_COMPPGA_Create_UserInit	Performs user-defined initialization relating to the comparator/programmable gain amplifier.
r_compn_interrupt	Performs processing in response to the comparator interrupt INTCMP*n*.
R_COMPn_Start	Begins comparison of reference input voltage and analog input voltage.
R_COMPn_Stop	Stops comparison of reference input voltage and analog input voltage.
R_PGA_Start	Starts the operation of programmable gain amplifier.
R_PGA_Stop	Ends the operation of programmable gain amplifier.
R_PWMOPT_Start	Supplies the clock to the 6-phase PWM option. In addition, sets the operation mode of the 6-phase PWM option.
R_PWMOPT_Stop	Halts the clock supplied to the 6-phase PWM option.

R_COMPPGA_Create

Performs initialization necessary to control the comparator/programmable gain amplifier.

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

[Syntax]

```
void        R_COMPPGA_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_COMPPGA_Set_PowerOff

Halts the clock supplied to the comparator/programmable gain amplifier.

Remark Calling this API function changes the comparator/programmable gain amplifier to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void        R_COMPPGA_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_COMPPGA_Create_UserInit

Performs user-defined initialization relating to the comparator/programmable gain amplifier.

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

[Syntax]

```
void        R_COMPPGA_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_comp*n*_interrupt

Performs processing in response to the comparator interrupt INTCMP*n*.

Remark This API function is called as the interrupt process corresponding to the comparator interrupt INTCMP*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_compn_interrupt ( void );
```

CC-RL Compiler

```
static void    __near r_compn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_COMP*n*_Start

Begins comparison of reference input voltage and analog input voltage.

[Syntax]

```
void        R_COMPn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_COMP*n*_Stop

Stops comparison of reference input voltage and analog input voltage.

[Syntax]

```
void        R_COMPn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_PGA_Start

Starts the operation of programmable gain amplifier.

[Syntax]

```
void        R_PGA_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA_Stop

Ends the operation of programmable gain amplifier.

[Syntax]

```
void        R_PGA_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PWMOPT_Start

Supplies the clock to the 6-phase PWM option.
In addition, sets the operation mode of the 6-phase PWM option.

[Syntax]

```
void        R_PWMOPT_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PWMOPT_Stop

Halts the clock supplied to the 6-phase PWM option.

[Syntax]

```
void          R_PWMOPT_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.3.29 Serial array unit

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

Table 3.30 API Functions: [Serial Array Unit] (1)

API Function Name	Function
R_SAUm_Create	Performs initialization necessary to control the serial array unit.
R_SAUm_Create_UserInit	Performs user-defined initialization related to the serial array unit.
R_SAUm_Reset	Reset the serial array unit.
R_SAUm_Set_PowerOff	Halts the clock supplied to the serial array unit.
R_SAUm_Set_SnoozeOn	Enables the switch from STOP mode to SNOOZE mode.
R_SAUm_Set_SnoozeOff	Disables the switch from STOP mode to SNOOZE mode.
R_UARTn_Create	Performs initialization necessary to perform the UART communication.
r_uartn_interrupt_send	Performs processing in response to the UART transmission end interrupt INTST*n*.
r_uartn_interrupt_receive	Performs processing in response to the UART reception end interrupt INTSR*n*.
r_uartn_callback_error	Performs processing in response to the reception error interrupt INTSRE*n*.
R_UARTn_Start	Sets UART communication to standby mode.
R_UARTn_Stop	Ends UART communication.
R_UARTn_Send	Starts UART data transmission.
R_UARTn_Receive	Starts UART data reception.
r_uartn_callback_sendend	Performs processing in response to the UART transmission end interrupt INTST*n*.
r_uartn_callback_receiveend	Performs processing in response to the UART reception end interrupt INTSR*n*.
r_uartn_callback_error	Performs processing in response to the UART reception error interrupt INTSRE*n*.
r_uartn_callback_softwareoverrun	Performs processing in response to detection of overrun error.
R_CSImn_Create	Performs initialization necessary to perform the 3-wire serial I/O communication.
r_csimn_interrupt	Performs processing in response to the CSI communication end interrupt INTCSI*mn*.
R_CSImn_Start	Sets 3-wire serial I/O communication to standby mode.
R_CSImn_Stop	Ends 3-wire serial I/O communication.
R_CSImn_Send	Starts CSI data transmission.
R_CSImn_Receive	Starts CSI data reception.
R_CSImn_Send_Receive	Starts CSI data transmission/reception.
r_csimn_callback_sendend	Performs processing in response to the CSI transmission end interrupt INTCSI*mn*.

Table 3.31 API Functions: [Serial Array Unit] (2)

API Function Name	Function
r_csimn_callback_receiveend	Performs processing in response to the CSI reception end interrupt INTCSI*mn*.
r_csimn_callback_error	Performs processing in response to the CSI reception error interrupt INTSRE*n*.
R_IICmn_Create	Performs initialization necessary to perform the simplified IIC communication.
r_iicmn_interrupt	Performs processing in response to the simple IIC communication end interrupt INTIIC*mn*.
R_IICmn_StartCondition	Generates start conditions.
R_IICmn_StopCondition	Generates stop conditions.
R_IICmn_Stop	Ends simplified IIC communication.
R_IICmn_Master_Send	Starts simple IIC master transmission.
R_IICmn_Master_Receive	Starts simple IIC master reception.
r_iicmn_callback_master_sendend	Performs processing in response to the simple IICmn master transmission end interrupt INTIIC*mn*.
r_iicmn_callback_master_receiveend	Performs processing in response to the simple IICmn master reception end interrupt INTIIC*mn*.
r_iicmn_callback_master_error	Performs processing in response to detection of parity error (ACK error).

R_SAU*m*_Create

Performs initialization necessary to control the serial array unit.

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

[Syntax]

```
void        R_SAUm_Create ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_SAU*m*_Create_UserInit

Performs user-defined initialization related to the serial array unit.

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

[Syntax]

```
void        R_SAUm_Create_UserInit ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_SAUm_Reset

Reset the serial array unit.

[Syntax]

```
void          R_SAUm_Reset ( void );
```

Remark   *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_SAU*m*_Set_PowerOff

Halts the clock supplied to the serial array unit.

Remark Calling this API function changes the serial array unit to reset status.
For this reason, writes to the control registers (e.g. serial clock select register *n*: SPS*n*) after this API function is called are ignored.

[Syntax]

```
void        R_SAUm_Set_PowerOff ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_SAU*m*_Set_SnoozeOn

Enables the switch from STOP mode to SNOOZE mode.

[Syntax]

```
void        R_SAUm_Set_SnoozeOn ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_SAU*m*_Set_SnoozeOff

Disables the switch from STOP mode to SNOOZE mode.

[Syntax]

```
void        R_SAUm_Set_SnoozeOff ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_UART*n*_Create

Performs initialization necessary to perform the UART communication.

Remark This API function is used as an internal function of R_SAUm_Create.
For this reason, there is normally no need to call it from a user program.

[Syntax]

```
void        R_UARTn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uart*n*_interrupt_send

Performs processing in response to the UART transmission end interrupt INTST*n*.

Remark This API function is called as the interrupt process corresponding to the UART transmission end interrupt INTST*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_uartn_interrupt_send ( void );
```

CC-RL Compiler

```
static void    __near r_uartn_interrupt_send ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uart*n*_interrupt_receive

Performs processing in response to the UART reception end interrupt INTSR*n*.

Remark This API function is called as the interrupt process corresponding to the UART reception end interrupt INTSR*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_uartn_interrupt_receive ( void );
```

CC-RL Compiler

```
static void     __near r_uartn_interrupt_receive ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uart*n*_interrupt_error

Performs processing in response to the reception error interrupt INTSRE*n*.

Remark This API function is called as the interrupt process corresponding to the reception error interrupt INTSRE*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_uartn_interrupt_error ( void );
```

CC-RL Compiler

```
static void    __near r_uartn_interrupt_error ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UART*n*_Start

Sets UART communication to standby mode.

[Syntax]

```
void        R_UARTn_Start ( void );
```

Remark   *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UART*n*_Stop

Ends UART communication.

[Syntax]

```
void          R_UARTn_Stop ( void );
```

Remark   *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UART*n*_Send

Starts UART data transmission.

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

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

[Syntax]

```
#include        "r_cg_macrodriver.h"
MD_STATUS    R_UARTn_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 specification

R_UARTn_Receive

Starts UART data reception.

Remark 1. This API function performs byte-level UART 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. Actual UART reception starts after this API function is called, and R_UARTn_Start is then called.

[Syntax]

```
#include        "r_cg_macrodriver.h"
MD_STATUS    R_UARTn_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 received data
I	uint16_t *rx_num*;	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

r_uart*n*_callback_sendend

Performs processing in response to the UART transmission end interrupt INTST*n*.

Remark This API function is called as the callback routine of interrupt process r_uartn_interrupt_send corresponding to the UART transmission end interrupt INTST*n* (performed when number of transmission data specified by R_UARTn_Send argument *tx_num* has been completed).

[Syntax]

```
static void     r_uartn_callback_sendend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uart*n*_callback_receiveend

Performs processing in response to the UART reception end interrupt INTSR*n*.

Remark This API function is called as the callback routine of interrupt process r_uartn_interrupt_receive corresponding to the UART transmission end interrupt INTSR*n* (performed when number of received data specified by R_UARTn_Receive argument *rx_num* has been completed).

[Syntax]

```
static void    r_uartn_callback_receiveend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uart*n*_callback_error

Performs processing in response to the UART reception error interrupt INTSRE*n*.

Remark This API function is called as the callback routine of interrupt process r_uartn_interrupt_error corresponding to the UART reception error interrupt INTSRE*n*.

[Syntax]

```
#include      "r_cg_macrodriver.h"
static void   r_uartn_callback_error ( uint8_t err_type );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	uint8_t *err_type*;	Trigger for UART reception error interrupt 00000xx1B: Overrun error 00000x1xB: Parity error 000001xxB: Framing error

[Return value]

None.

r_uart*n*_callback_softwareoverrun

Performs processing in response to detection of overrun error.

Remark This API function is called as the callback routine of interrupt process r_uartn_interrupt_receive corresponding to the UART reception end interrupt INTSR*n* (process performed when the amount of data received is greater than the argument *rx_num* specified for R_UARTn_Receive).

[Syntax]

```
#include      "r_cg_macrodriver.h"
static void   r_uartn_callback_softwareoverrun ( uint16_t rx_data );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	uint16_t *rx_data*;	Receive data (greater than the argument *rx_num* specified for R_UARTn_Receive)

[Return value]

None.

R_CSI*mn*_Create

Performs initialization necessary to perform the 3-wire serial I/O communication.

Remark This API function is used as an internal function of R_SAUm_Create .
For this reason, there is normally no need to call it from a user program.

[Syntax]

```
void        R_CSImn_Create ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_csi*mn*_interrupt

Performs processing in response to the CSI communication end interrupt INTCSI*mn*.

Remark This API function is called as the interrupt process corresponding to the CSI communication end interrupt INTCSI*mn*.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_csimn_interrupt ( void );
```

CC-RL Compiler

```
static void    __near r_csimn_interrupt ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CSI*mn*_Start

Sets 3-wire serial I/O communication to standby mode.

[Syntax]

```
void        R_CSImn_Start ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CSI*mn*_Stop

Ends 3-wire serial I/O communication.

[Syntax]

```
void        R_CSImn_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_CSI*mn*_Send

Starts CSI data transmission.

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

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

[Syntax]

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

Remark *m* is the unit number, and *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 specification

R_CSI*mn*_Receive

Starts CSI data reception.

Remark 1. This API function performs byte-level CSI 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 CSI reception, R_CSImn_Start must be called before this API function is called.

[Syntax]

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

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

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

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_CSI*mn*_Send_Receive

Starts CSI data transmission/reception.

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

Remark 2. This API function performs byte-level CSI reception the number of times specified by the argument *tx_num*, and stores the data in the buffer specified by the argument *rx_buf*.

Remark 3. When performing a CSI reception, R_CSImn_Start must be called before this API function is called.

[Syntax]

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

Remark *m* is the unit number, and *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/receive
O	uint8_t * const *rx_buf*;	Pointer to a buffer to store the received data

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

r_csi*mn*_callback_sendend

Performs processing in response to the CSI transmission end interrupt INTCSI*mn*.

Remark 1. This API function is called as the callback routine of interrupt process r_csimn_interrupt corresponding to the CSI transmission end interrupt INTCSI*mn* (performed when number of transmission data specified by R_CSImn_Send or R_CSImn_Send_Receive argument *tx_num* has been completed).

Remark 2. If you repeat sending or receiving in countinous mode, add the following to the callback function in order to reset to countinous mode.

- r_csimn_callback_sendend (): “| = _0001_SAU_BUFFER_EMPTY;”

[Syntax]

```
static void    r_csimn_callback_sendend ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_csi*mn*_callback_receiveend

Performs processing in response to the CSI reception end interrupt INTCSI*mn*.

Remark 1. This API function is called as the callback routine of interrupt process r_csimn_interrupt corresponding to the CSI reception end interrupt INTCSI*mn* (performed when number of received data specified by R_CSImn_Receive or R_CSImn_Send_Receive argument *rx_num* has been completed).

Remark 2. If you repeat sending or receiving in countinous mode, add the following to the callback function in order to reset to countinous mode.

- r_csimn_callback_receiveend(): "SMRmn |= _0001_SAU_BUFFER_EMPTY;"

[Syntax]

```
static void    r_csin_callback_receiveend ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_csi*mn*_callback_error

Performs processing in response to the CSI reception error interrupt INTSRE*n*.

Remark This API function is called as the callback routine of interrupt process r_uartn_interrupt_error corresponding to the CSI reception error interrupt INTSRE*n*.

[Syntax]

```
#include        "r_cg_macrodriver.h"
static void     r_csimn_callback_error ( uint8_t err_type );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	uint8_t *err_type*;	Trigger for CSI reception error interrupt 00000xx1B: Overrun error

[Return value]

None.

R_IIC*mn*_Create

Performs initialization necessary to perform the simplified IIC communication.

Remark This API function is used as an internal function of R_SAUm_Create .
For this reason, there is normally no need to call it from a user program.

[Syntax]

```
void        R_IICIn_Create ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_iic*mn*_interrupt

Performs processing in response to the simple IIC transfer end interrupt INTIIC*mn*.

Remark 1. This API function is called as the interrupt process corresponding to the simple IIC transfer end interrupt INTIIC*mn*.

Remark 2. Stop condition is not generated in this API.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_iicmn_interrupt ( void );
```

CC-RL Compiler

```
static void     __near r_iicmn_interrupt ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IIC*mn*_StartCondition

Generates start conditions.

Remark This API function is used as an internal function of R_IICmn_Master_Send and R_IICmn_Master_Receive .
For this reason, there is normally no need to call it from a user program.

[Syntax]

```
void        R_IICmn_StartCondition ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IIC*mn*_StopCondition

Generates stop conditions.

Remark 1. User should manually check slave process completion and set stop condition in main() function.

Remark 2. Please avoid to set stop condition in r_iic*mn*_interrupt, r_iic*mn*_callback_master_sendend, r_iic*mn*_callback_master_receiveend and r_iic*mn*_callback_master_error.

[Syntax]

```
void        R_IICmn_StopCondition ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IIC*mn*_Stop

Ends simple IIC communication.

[Syntax]

```
void        R_IICmn_Stop ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IIC*mn*_Master_Send

Starts simple IIC master transmission.

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

Remark 2. Before calling this API, please check that communiacation is stopped/suspended and SDA/SCL are High level.

[Syntax]

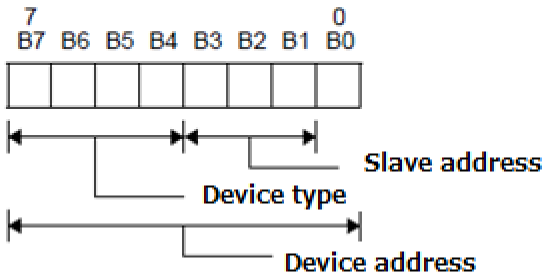
```
#include   "r_cg_macrodriver.h"
void       R_IICmn_Master_Send ( uint8_t adr, uint8_t * const tx_buf, uint16_t tx_num );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint8_t *adr*;	Device 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

Remark Below is shown the format for specifying device address *adr*.



[Return value]

None.

R_IIC*mn*_Master_Receive

Starts simple IIC master reception.

Remark 1. This API function performs byte-level simple IIC master 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. Before calling this API, please check that communiacation is stopped/suspended and SDA/SCL are High level.

[Syntax]

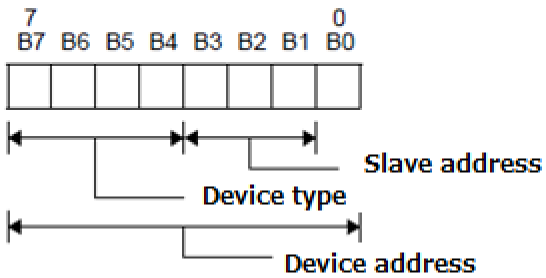
```
#include   "r_cg_macrodriver.h"
void       R_IICmn_Master_Receive ( uint8_t adr, uint8_t * const rx_buf, uint16_t rx_num );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

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

Remark Below is shown the format for specifying device address *adr*.



[Return value]

None.

r_iic*mn*_callback_master_sendend

Performs processing in response to the simple IICmn transfer end (master transmission end) interrupt INTIIC*mn*.

Remark This API function is called as the callback routine of interrupt process r_iicmn_interrupt corresponding to the simple IIC*mn* transfer end (master transmission end) interrupt INTIICmn (performed when number of transmission data specified by R_IICmn_Master_Send argument *tx_num* has been completed).

[Syntax]

```
static void    r_iicmn_callback_master_sendend ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_iic*mn*_callback_master_receiveend

Performs processing in response to the simple IICmn transfer end (master reception end) interrupt INTIIC*mn*.

Remark This API function is called as the callback routine of interrupt process r_iicmn_interrupt corresponding to the simple IICmn transfer end (master reception end) interrupt INTIIC*mn* (performed when number of received data specified by R_IICmn_Master_Receive argument *rx_num* has been completed).

[Syntax]

```
static void    r_iicn_callback_master_receiveend ( void );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_iic*mn*_callback_master_error

Performs processing in response to detection of ACK error or Overrun error.

Remark This API function is called as the callback routine of interrupt process r_iic*mn*_interrupt corresponding to the simple IIC transfer end interrupt INTIIC*mn*.

[Syntax]

```
#include      "r_cg_macrodriver.h"
static void   r_iicmn_callback_master_error ( MD_STATUS flag );
```

Remark *m* is the unit number, and *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	MD_STATUS *flag*;	Cause of communication error MD_NACK: Acknowledge not detected MD_OVERRUN: Overrun detected

[Argument(s)]

None.

[Return value]

None.

Usage example

Receive 4 bytes data by UART and transmit the received data as they are. Stop UART after transmission is finished.

[GUI setting example]

Serial				Used
	SAU0			Used
		Channel0		
			Channel 0	UART0 (Transmission/Receive)
			Data length setting (Receive)	8 bits
			Trasfer direction setting (Receive)	LSB
			Parity setting (Receive)	None
			Stop bit length setting (Receive)	1 bit fixed
			Receive data level setting	Normal
			Transfer rate setting (Receive)	9600(bps) (error:+0.16% Minimum permissible value:-5.17% Maximum permissible value:+5.16%)
			Reception end interrupt priority (INTSR0)	Low
			Reception end (Callback function setting)	Used
			Reception error (Callback function setting)	Used
			Transfer mode setting	Single transfer mode
			Data length setting (Transmit)	8 bits
			Transfer direction setting (Transmit)	LSB
			Parity setting (Transmit)	None
			Stop bit length setting (Transmit)	1 bit
			Transmit data level setting	Normal
			Transfer rate setting (Transmit)	9600(bps) (error: +0.16%)
			Transmit end interrupt priority (INTST0)	Low
			Transmission end (Callback function setting)	Used

[API setting example]

r_main.c

```
/* Start user code for global. Do not edit comment generated here */
extern volatile uint8_t g_uart0_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 UART0 module operation */
        R_UART0_Start();

        /* Receive UART0 data */
        R_UART0_Receive((uint8_t *)g_uart0_buf, 4U);

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

r_cg_serial_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_uart0_buf[4];
/* End user code. Do not edit comment generated here */

static void r_uart0_callback_receiveend(void)
{
        /* Start user code. Do not edit comment generated here */
        /* Send UART0 data */
        R_UART0_Send((uint8_t *)g_uart0_buf, 4U);
        /* End user code. Do not edit comment generated here */
}

static void r_uart0_callback_sendend(void)
{
        /* Start user code. Do not edit comment generated here */
        /* Stop the UART0 module operation */
        R_UART0_Stop();
        /* End user code. Do not edit comment generated here */
}
```

3.3.30 Serial array unit 4

Below is a list of API functions output by the Code Generator for serial array unit 4 (DALI/UART4) use.

Table 3.32 API Functions: [Serial Array Unit 4]

API Function Name	Function
R_DALIn_Create	Performs initialization necessary to control the serial array unit 4 (DALI/ UART4).
r_dalin_interrupt_send	Performs processing in response to the DALI transmission end interrupt INTSTDL*n*.
r_dalin_interrupt_receive	Performs processing in response to the DALI reception end interrupt INTSRDL*n*.
r_dalin_interrupt_error	Performs processing in response to the DALI reception error interrupt INTSREDL*n*.
R_DALIn_Start	Sets DALI communication to standby mode.
R_DALIn_Stop	Ends DALI communication.
R_DALIn_Send	Starts DALI data transmission.
R_DALIn_Receive	Starts DALI data reception.
r_dalin_callback_sendend	Performs processing in response to the DALI transmission end interrupt INTSTDL*n*.
r_dalin_callback_receiveend	Performs processing in response to the DALI reception end interrupt INTSRDL*n*.
r_dalin_callback_error	Performs processing in response to the DALI reception error interrupt INTSREDL*n*.
r_dalin_callback_softwareoverrun	Performs processing in response to detection of overrun error.

R_DALI*n*_Create

Performs initialization necessary to control the serial array unit 4 (DALI/UART4).

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

[Syntax]

```
void R_DALIn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_dali*n*_interrupt_send

Performs processing in response to the DALI transmission end interrupt INTSTDL*n*.

Remark This API function is called as the interrupt process corresponding to the DALI transmission end interrupt INTSTDL*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_dalin_interrupt_send ( void );
```

CC-RL Compiler

```
static void     __near r_dalin_interrupt_send ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_dali*n*_interrupt_receive

Performs processing in response to the DALI reception end interrupt INTSRDL*n*.

Remark This API function is called as the interrupt process corresponding to the DALI reception end interrupt INTSRDL*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_dalin_interrupt_receive ( void );
```

CC-RL Compiler

```
static void     __near r_dalin_interrupt_receive ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_dali*n*_interrupt_error

Performs processing in response to the DALI reception error interrupt INTSREDL*n*.

Remark This API function is called as the interrupt process corresponding to the DALI reception error interrupt INTSREDL*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_dalin_interrupt_error ( void );
```

CC-RL Compiler

```
static void    __near r_dalin_interrupt_error ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DALI*n*_Start

Sets DALI communication to standby mode.

[Syntax]

```
void        R_DALIn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DALI*n*_Stop

Ends DALI communication.

[Syntax]

```
void        R_DALIn_Stop ( void );
```

Remark   *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DALI*n*_Send

Starts DALI data transmission.

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

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

[Syntax]

```
#include        "r_cg_macrodriver.h"
MD_STATUS    R_DALIn_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 specification

R_DALI*n*_Receive

Starts DALI data reception.

Remark 1. This API function performs byte-level DALI 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. Actual DALI reception starts after this API function is called, and R_DALIn_Start is then called.

[Syntax]

```
#include        "r_cg_macrodriver.h"
MD_STATUS       R_DALIn_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 received data
I	uint16_t *rx_num*;	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

r_dali*n*_callback_sendend

Performs processing in response to the DALI transmission end interrupt INTSTDL*n*.

Remark This API function is called as the callback routine of interrupt process r_dalin_interrupt_send corresponding to the DALI transmission end interrupt INTSTDL*n* (performed when number of transmission data specified by R_DALIn_Send rgument *tx_num* has been completed).

[Syntax]

```
static void    r_dalin_callback_sendend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_dali*n*_callback_receiveend

Performs processing in response to the DALI reception end interrupt INTSRDL*n*.

Remark This API function is called as the callback routine of interrupt process r_dalin_interrupt_receive corresponding to the DALI reception end interrupt INTSRDL*n* (performed when number of received data specified by R_DALIn_Receive argument *rx_num* has been completed).

[Syntax]

```
static void     r_dalin_callback_receiveend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_dali*n*_callback_error

Performs processing in response to the DALI reception error interrupt INTSREDL*n*.

Remark This API function is called as the callback routine of interrupt process r_dali*n*_interrupt_error corresponding to the DALI reception error interrupt INTSREDL*n*.

[Syntax]

```
#include      "r_cg_macrodriver.h"
static void   r_dalin_callback_error ( uint8_t err_type );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	uint8_t *err_type*;	Trigger for DALI reception error interrupt 00000xx1B: Overrun error 00000x1xB: Parity error 000001xxB: Framing error

[Return value]

None.

r_dali*n*_callback_softwareoverrun

Performs processing in response to detection of overrun error.

Remark This API function is called as the callback routine of interrupt process r_dalin_interrupt_receive corresponding to the DALI reception end interrupt INTSRDLn (process performed when the amount of data received is greater than the argument rx_num specified for R_DALIn_Receive .

[Syntax]

```
#include      "r_cg_macrodriver.h"
static void   r_dalin_callback_softwareoverrun ( uint16_t rx_data );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	uint16_t *rx_data*;	Receive data (greater than the argument *rx_num* specified for R_DALIn_Receive)

[Return value]

None.

Usage example

Please refer to 'Usage example in 3.2.30 Serial Array Unit' when using this module as UART.

3.3.31 Asynchronous serial interface LIN-UART

Below is a list of API functions output by the Code Generator for asynchronous serial interface LIN-UART (UARTF) use.

Table 3.33 API Functions: [Asynchronous Serial Interface LIN-UART]

API Function Name	Function
R_UARTFn_Create	Performs initialization necessary to control the asynchronous serial interface LIN-UART (UARTF).
R_UARTFn_Create_UserInit	Performs user-defined initialization related to the asynchronous serial interface LIN-UART (UARTF).
r_uartfn_interrupt_send	Performs processing in response to the LIN-UART transmission end interrupt INTLT.
r_uartfn_interrupt_receive	Performs processing in response to the LIN-UART reception end interrupt INTLR.
r_uartfn_interrupt_error	Performs processing in response to the LIN-UART reception status interrupt INTLS.
R_UARTFn_Start	Sets LIN communication to standby mode.
R_UARTFn_Stop	Ends LIN communication.
R_UARTFn_Set_PowerOff	Halts the clock supplied to the asynchronous serial interface LIN-UART (UARTF).
R_UARTFn_Send	Starts UARTF data transmission.
R_UARTFn_Receive	Starts UARTF data reception.
R_UARTFn_Set_DataComparisonOn	Starts the data comparison.
R_UARTFn_Set_DataComparisonOff	Ends the data comparison.
r_uartfn_callback_sendend	Performs processing in response to the LIN-UART transmission end interrupt INTLT.
r_uartfn_callback_receiveend	Performs processing in response to the LIN-UART reception end interrupt INTLR.
r_uartfn_callback_error	Performs processing in response to the LIN-UART reception status interrupt INTLS.
r_uartfn_callback_softwareoverrun	Performs processing in response to detection of overrun error.
r_uartfn_callback_expbitdetect	Performs processing in response to detection of expansion bit.
r_uartfn_callback_idmatch	Performs processing in response to match of ID parity.

R_UARTF*n*_Create

Performs initialization necessary to control the asynchronous serial interface LIN-UART (UARTF).

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

[Syntax]

```
void        R_UARTFn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTF*n*_Create_UserInit

Performs user-defined initialization related to the asynchronous serial interface LIN-UART (UARTF).

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

[Syntax]

```
void        R_UARTFn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartf*n*_interrupt_send

Performs processing in response to the LIN-UART transmission end interrupt INTLT.

Remark This API function is called as the interrupt process corresponding to the LIN-UART transmission end interrupt INTLT.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_uartfn_interrupt_send ( void );
```

CC-RL Compiler

```
static void     __near r_uartfn_interrupt_send ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartf*n*_interrupt_receive

Performs processing in response to the LIN-UART reception end interrupt INTLR.

Remark This API function is called as the interrupt process corresponding to the LIN-UART reception end interrupt INTLR.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_uartfn_interrupt_receive ( void );
```

CC-RL Compiler

```
static void     __near r_uartfn_interrupt_receive ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartf*n*_interrupt_error

Performs processing in response to the LIN-UART reception status interrupt INTLS.

Remark This API function is called as the interrupt process corresponding to the LIN-UART reception status interrupt INTLS.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_uartfn_interrupt_error ( void );
```

CC-RL Compiler

```
static void     __near r_uartfn_interrupt_error ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTF*n*_Start

Sets LIN communication to standby mode.

[Syntax]

```
void        R_UARTFn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTF*n*_Stop

Ends LIN communication.

[Syntax]

```
void        R_UARTFn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTF*n*_Set_PowerOff

Halts the clock supplied to the asynchronous serial interface LIN-UART (UARTF).

Remark Calling this API function changes the asynchronous serial interface LIN-UART (UARTF) to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void        R_UARTFn_Set_PowerOff ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTF*n*_Send

Starts UARTF data transmission.

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

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

Remark 2. If the asynchronous serial interface LIN-UART (UARTF) is used in expansion bit mode, then store the data to send in the buffer specified by argument *tx_buf*, in the following format.
"8-bit data", "Expansion bit", "8-bit data", "Expansion bit", ...

[Syntax]

```
#include        "r_cg_macrodriver.h"
MD_STATUS       R_UARTFn_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 specification
MD_DATAEXISTS	Executing transmission process

R_UARTF*n*_Receive

Starts UARTF data reception.

Remark 1. This API function performs byte-level UARTF 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. Actual UARTF reception starts after this API function is called, and R_UARTFn_Start is then called.

Remark 3. If the asynchronous serial interface LIN-UART (UARTF) is used in expansion bit mode, then the received data is stored in the buffer specified by argument *rx_buf*, in the following format.
"8-bit data", "Expansion bit", "8-bit data", "Expansion bit", ...

[Syntax]

```
#include        "r_cg_macrodriver.h"
MD_STATUS       R_UARTF_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 received data
I	uint16_t *rx_num*;	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_UARTF*n*_Set_DataComparisonOn

Starts the data comparison.

Remark Calling this API function switches the asynchronous serial interface LIN-UART (UARTF) to expansion bit mode (with data comparison).

[Syntax]

```
void    R_UARTFn_Set_DataComparisonOn ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTF*n*_Set_DataComparisonOff

Ends the data comparison.

Remark Calling this API function switches the asynchronous serial interface LIN-UART (UARTF) to expansion bit mode (with no data comparison).

[Syntax]

```
void    R_UARTFn_Set_DataComparisonOff ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartf*n*_callback_sendend

Performs processing in response to the LIN-UART transmission end interrupt INTLT.

Remark This API function is called as the callback routine of interrupt process r_uartfn_interrupt_send corresponding to the LIN-UART transmission end interrupt INTLT (performed when number of transmission data specified by R_UARTFn_Send argument *tx_num* has been completed).

[Syntax]

```
static void    r_uartfn_callback_sendend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartf*n*_callback_receiveend

Performs processing in response to the LIN-UART reception end interrupt INTLR.

Remark This API function is called as the callback routine of interrupt process r_uartfn_interrupt_receive corresponding to the LIN-UART reception end interrupt INTLR (performed when number of received data specified by R_UARTFn_Receive argument *rx_num* has been completed).

[Syntax]

```
static void    r_uartfn_callback_receiveend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartfn_callback_error

Performs processing in response to the LIN-UART reception status interrupt INTLS.

Remark This API function is called as the callback routine of interrupt process r_uartfn_interrupt_error corresponding to the LIN-UART reception status interrupt INTLS.

[Syntax]

```
static void    r_uartfn_callback_error ( uint8_t err_type );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	uint8_t *err_type*;	Trigger for LIN-UART reception status interrupt 00000xx1B: Overrun error 00000x1xB: Parity error 000001xxB: Framing error

[Return value]

None.

r_uartf*n*_callback_softwareoverrun

Performs processing in response to detection of overrun error.

Remark This API function is called as the callback routine of interrupt process r_uartfn_interrupt_receive corresponding to the LIN-UART reception end interrupt INTLR (performed when number of received data specified by R_UARTFn_Receive argument *rx_num* has been completed).

[Syntax]

```
static void    r_uartfn_callback_softwareoverrun ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartf*n*_callback_expbitdetect

Performs processing in response to detection of expansion bit.

Remark This API function is called as the callback routine of interrupt process r_uartfn_interrupt_error corresponding to the LIN-UART reception status interrupt INTLS (performed when expansion bit has been detected).

[Syntax]

```
static void    r_uartfn_callback_expbitdetect ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartf*n*_callback_idmatch

Performs processing in response to match ID parity.

Remark This API function is called as the callback routine of interrupt process r_uartfn_interrupt_error corresponding to the LIN-UART reception status interrupt INTLS (performed when ID parity has been matched).

[Syntax]

```
static void    r_uartfn_callback_idmatch ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

Usage example

Please refer to 'Usage example in 3.2.30 Serial Array Unit' when using this module as UART.

3.3.32 Serial interface IICA

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

Table 3.34 API Functions: [Serial Interface IICA]

API Function Name	Function
R_IICAn_Create	Performs initialization necessary to control the serial interface IICA.
R_IICAn_Create_UserInit	Performs user-defined initialization related to the serial interface IICA.
r_iican_interrupt	Performs processing in response to the IICA communication end interrupt INTIICA*n*.
R_IICAn_StopCondition	Generates stop conditions.
R_IICAn_Stop	Ends IICA communication.
R_IICAn_Reset	Reset the serial interface IICA.
R_IICAn_Set_PowerOff	Halts the clock supplied to the serial interface IICA.
R_IICAn_Master_Send	Starts IICA master transmission.
R_IICAn_Master_Receive	Starts IICA master reception.
r_iican_callback_master_sendend	Performs processing in response to the IICA master transmission end interrupt INTIICA*n*.
r_iican_callback_master_receiveend	Performs processing in response to the IICA master reception end interrupt INTIICA*n*.
r_iican_callback_master_error	Performs processing in response to detection of IICA master communication error.
R_IICAn_Slave_Send	Starts IICA slave transmission.
R_IICAn_Slave_Receive	Starts IICA slave reception.
r_iican_callback_slave_sendend	Performs processing in response to the IICA slave transmission end interrupt INTIICA*n*.
r_iican_callback_slave_receiveend	Performs processing in response to the IICA slave reception end interrupt INTIICA*n*.
r_iican_callback_slave_error	Performs processing in response to detection of IICA slave communication error.
r_iican_callback_getstopcondition	Performs processing in response to detection of stop condition.
R_IICAn_Set_SnoozeOn	Enables operation of the address match wakeup function in STOP mode.
R_IICAn_Set_SnoozeOff	Disables operation of the address match wakeup function in STOP mode.
R_IICAn_Set_WakeupOn	Enables operation of the address match wakeup function in STOP mode.
R_IICAn_Set_WakeupOff	Disables operation of the address match wakeup function in STOP mode.

R_IICA*n*_Create

Performs initialization necessary to control the serial interface IICA.

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

[Syntax]

```
void        R_IICAn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_Create_UserInit

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

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

[Syntax]

```
void R_IICAn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_iica*n*_interrupt

I Performs processing in response to the IICA communication end interrupt INTIICA*n*.

Remark This API function is called as the interrupt process corresponding to the IICA communication end interrupt INTIICA*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_iican_interrupt ( void );
```

CC-RL Compiler

```
static void     __near r_iican_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_StopCondition

Generates stop conditions.

Remark After calling this API function, please confirm a detection of stop condition by SPD0 bit before stopping IICA.

[Syntax]

```
void        R_IICAn_StopCondition ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_Stop

Ends IICA communication.

[Syntax]

```
void        R_IICAn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_Reset

Reset the serial interface IICA.

[Syntax]

```
void        R_IICAn_Reset ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_Set_PowerOff

Halts the clock supplied to the serial interface IICA.

Remark Calling this API function changes the serial interface IICA to reset status.
For this reason, writes to the control registers (e.g. IICA control register n: IICCTL*n*) after this API function is called are ignored.

[Syntax]

```
void        R_IICAn_Set_PowerOff ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_Master_Send

Starts IICA master transmission.

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

[Syntax]

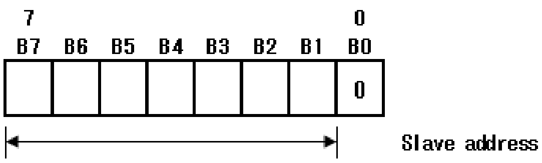
```
#include        "r_cg_macrodriver.h"
MD_STATUS       R_IICAn_Master_Send ( uint8_t adr, uint8_t * const tx_buf,
  uint16_t tx_num, uint8_t wait );
```

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
I	uint8_t *wait*	Setup time of start conditions

Remark Syntax of slave assress *adr* is as below.
Specify the slave address in the upper 7 bits. Set the least significant bit to 0 in this API function.



[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Bus communication status
MD_ERROR2	Start condition is not detected

R_IICA*n*_Master_Receive

Starts IICA master reception.

Remark This API function performs byte-level IICA master reception the number of times specified by the argument *rx_num*, and stores the data in the buffer specified by the argument *rx_buf*.

[Syntax]

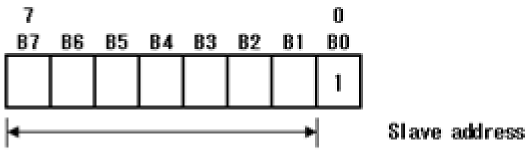
```
#include        "r_cg_macrodriver.h"
MD_STATUD    R_IICAn_Master_Receive ( uint8_t adr, uint8_t * const rx_buf,
 uint16_t rx_num, uint8_t wait );
```

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 received data
I	uint16_t *rx_num*;	Total amount of data to receive
I	uint8_t *wait*	Setup time of start conditions

Remark Syntax of slave assress *adr* is as below.
Specify the slave address in the upper 7 bits. Set the least significant bit to 1 in this API function.



[Return value]

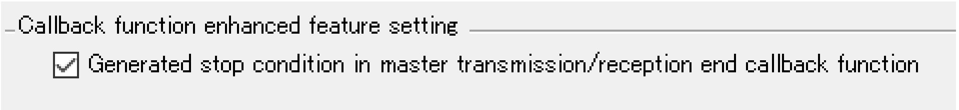
Macro	Description
MD_OK	Normal completion
MD_ERROR1	Bus communication status
MD_ERROR2	Start condition is not detected

r_iica*n*_callback_master_sendend

Performs processing in response to the IICA master transmission end interrupt INTIICA*n*.

Remark 1. This API function is called as the callback routine of interrupt process r_iican_interrupt corresponding to the IICA master transmission end interrupt INTIICA*n*.

Remark 2. Whether or not R_IICAn_StopCondition is generated into this API function depends on the IICA GUI settings.



Checked: R_IICAn_StopCondition is generated in this API
Unchecked: R_IICAn_StopCondition is not generated in this API

[Syntax]

```
static void    r_iican_callback_master_sendend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

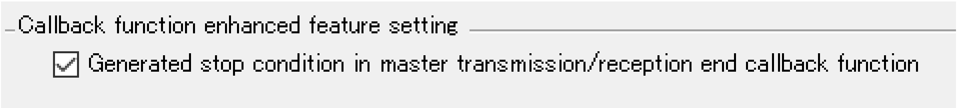
None.

r_iican_callback_master_receiveend

Performs processing in response to the IICA master reception end interrupt INTIICA*n*.

Remark 1. This API function is called as the callback routine of interrupt process r_iican_interrupt corresponding to the IICA master reception end interrupt INTIICA*n*.

Remark 2. Whether or not R_IICAn_StopCondition is generated into this API function depends on the IICA GUI settings.



Checked: R_IICAn_StopCondition is generated in this API
Unchecked: R_IICAn_StopCondition is not generated in this API

[Syntax]

```
static void    r_iican_callback_master_receiveend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_iica*n*_callback_master_error

Performs processing in response to detection of IICA master communication error.

[Syntax]

```
#include      "r_cg_macrodriver.h"
static void   r_iican_callback_master_error ( MD_STATUS flag );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	MD_STATUS *flag*;	Cause of communication error MD_SPT: Stop condition detected MD_NACK: Acknowledge not detected (No slave that matches the address/A slave can receive no more data or does not require the next data)

[Return value]

None.

R_IICA*n*_Slave_Send

Starts IICA slave transmission.

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

[Syntax]

```
#include    "r_cg_macrodriver.h"
void        R_IICAn_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]

None.

R_IICA*n*_Slave_Receive

Starts IICA slave reception.

Remark This API function performs byte-level IICA 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*.

[Syntax]

```
#include   "r_cg_macrodriver.h"
void       R_IICAn_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 received data
I	uint16_t *rx_num*;	Total amount of data to receive

[Return value]

None.

r_iica*n*_callback_slave_sendend

Performs processing in response to the IICA slave transmission end interrupt INTIICA*n*.

Remark This API function is called as the callback routine of interrupt process r_iican_interrupt corresponding to the IICA slave transmission end interrupt INTIICA*n*.

[Syntax]

```
static void    r_iican_callback_master_sendend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_iica*n*_callback_slave_receiveend

Performs processing in response to the IICA slave reception end interrupt INTIICA*n*.

Remark This API function is called as the callback routine of interrupt process r_iican_interrupt corresponding to the IICA slave reception end interrupt INTIICA*n*.

[Syntax]

```
static void     r_iican_callback_slave_receiveend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_iica*n*_callback_slave_error

Performs processing in response to detection of IICA slave communication error.

[Syntax]

```
#include      "r_cg_macrodriver.h"
static void   r_iican_callback_slave_error ( MD_STATUS flag );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	MD_STATUS *flag*;	Cause of communication error MD_ERROR: Address mismatch detected MD_NACK: Acknowledge not detected (Master receiving end)

[Return value]

None.

r_iica*n*_callback_getstopcondition

Performs processing in response to detection of stop condition.

[Syntax]

```
static void    r_iican_callback_getstopcondition ( void );
```

Remark   *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_Set_SnoozeOn

Enables operation of the address match wakeup function in STOP mode.

[Syntax]

```
void        R_IICAn_Set_SnoozeOn ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_Set_SnoozeOff

Disables operation of the address match wakeup function in STOP mode.

[Syntax]

```
void        R_IICAn_Set_SnoozeOff ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_Set_WakeupOn

Enables operation of the address match wakeup function in STOP mode.

[Syntax]

```
void        R_IICAn_Set_WakeupOn ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_IICA*n*_Set_WakeupOff

Disables operation of the address match wakeup function in STOP mode.

[Syntax]

```
void        R_IICAn_Set_WakeupOff ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

Usage example (Master, Transmit)

Transmit 4Bytes data by Master.

[GUI setting example]

Serial				Used
	IICA0			Used
			Transfer mode	Single master
		Master0		
			Clock mode setting	fCLK/2
			Local address setting	16
			Operation mode setting	Normal
			Transfer clock (fSCL)	100000(bps)(Actual value：99378.882)
			Communication endinterrupt priority (INTIICA0)	High
			Master transmission end (Callback function setting)	Used
			Master reception end (Callback function setting)	Unused
			Master error (Callback function setting)	Unused
			Generates stop condition in master transmission/reception end callback function (Callback function enhanced feature setting)	Used

[API setting example]

r_main.c

```
/* Start user code for pragma. Do not edit comment generated here */
#define SLAVE_ADDR (0xA0)       /* slave address */
/* End user code. Do not edit comment generated here */

/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_iica0_tx_buf[4] = { 'A', 'B', 'C', 'D' };
/* 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 to send data as master mode */
        R_IICA0_Master_Send(SLAVE_ADDR, (uint8_t *)g_iica0_tx_buf, 4U, 128U);

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

r_cg_serial_user.c

```
static void r_iica0_callback_master_sendend(void)
{
        SPT0 = 1U;
        /* Start user code. Do not edit comment generated here */
        /* Stop IICA0 module operation */
        R_IICA0_Stop();
        /* End user code. Do not edit comment generated here */
}
```

Usage example (Slave, Reception)

Receive 4Bytes data by Slave.

[GUI setting example]

Serial				Used
	SAU0			Unused
	SAU1			Unused
	IICA0			Used
			Transfer mode	Slave
		Slave0		
			Clock mode setting	fCLK/2
			Local address setting	0xA0
			Operation mode setting	Normal
			Wakeup function setting	Off
			Communication endinterrupt priority (INTIICA0)	Low
			Slave transmission end (Callback function setting)	Unused
			Slave reception end (Callback function setting)	Used
			Slave error (Callback function setting)	Unused

[API setting example]

r_main.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_iica0_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 */
	/* Receive data as slave mode */
	R_IICA0_Slave_Receive((uint8_t *)g_iica0_rx_buf, 4U);

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

r_cg_serial_user.c

```
static void r_iica0_callback_slave_receiveend(void)
{
	/* Start user code. Do not edit comment generated here */
	/* Stop IICA0 module operation */
	R_IICA0_Stop();
	/* End user code. Do not edit comment generated here */
}
```

Usage example (Master, Reception)

Receive 4Bytes data by Master.

[GUI setting example]

Serial				Used
	IICA0			Used
			Transfer mode	Single master
		Master0		
			Clock mode setting	fCLK/2
			Local address setting	16
			Operation mode setting	Normal
			Transfer clock (fSCL)	100000(bps)(Actual value：99378.882)
			Communication endinterrupt priority (INTIICA0)	Low
			Master transmission end (Callback function setting)	Unused
			Master reception end (Callback function setting)	Used
			Master error (Callback function setting)	Unused
			Generates stop condition in master transmission/reception end callback function (Callback function enhanced feature setting)	Used

[API setting example]

r_main.c

```
/* Start user code for pragma. Do not edit comment generated here */
#define SLAVE_ADDR (0xA0)       /* slave address */
/* End user code. Do not edit comment generated here */

/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_iica0_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 to receive IICA0 data as master mode */
        R_IICA0_Master_Receive(SLAVE_ADDR, (uint8_t *)g_iica0_rx_buf, 4U, 128U);

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

r_cg_serial_user.c

```
static void r_iica0_callback_master_receiveend(void)
{
        SPT0 = 1U;
        /* Start user code. Do not edit comment generated here */
        /* Stop IICA0 module operation */
        R_IICA0_Stop();
        /* End user code. Do not edit comment generated here */
}
```

Usage example (Slave, Transmit)

Transmit 4Bytes data by Slave.

[GUI setting example]

Serial				Used
	IICA0			Used
			Transfer mode	Slave
		Slave0		
			Clock mode setting	fCLK/2
			Local address setting	0xA0
			Operation mode setting	Normal
			Wakeup function setting	Off
			Communication endinterrupt priority (INTIICA0)	Low
			Slave transmission end (Callback function setting)	Used
			Slave reception end (Callback function setting)	Unused

[API setting example]

r_main.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_iica0_tx_buf[4] = { 'A', 'B', 'C', 'D' };
/* End user code. Do not edit comment generated here */

void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Send data as slave mode */
    R_IICA0_Slave_Send((uint8_t *)g_iica0_tx_buf, 4U);

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

r_cg_serial_user.c

```
static void r_iica0_callback_slave_sendend(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop IICA0 module operation */
    R_IICA0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

3.3.33 LCD controller/driver

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

Table 3.35 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_interrupt	Performs processing in response to the LCD frame interrupt INTLCD.
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_Set_VoltageOn	Enables operation of internal voltage boost circuit and capacitor split circuit.
R_LCD_Set_VoltageOff	Disables operation of internal voltage boost circuit and capacitor split circuit.
R_LCD_Set_PowerOff	Halts the clock supplied to the LCD controller/driver.
R_LCD_VoltageOn	Enables operation of internal voltage boost circuit and capacitor split circuit.
R_LCD_VoltageOff	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_interrupt

Performs processing in response to the LCD frame interrupt INTLCD.

Remark This API function is called as the interrupt process corresponding to the LCD frame interrupt INTLCD.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_lcd_interrupt ( void );
```

CC-RL Compiler

```
static void    __near r_lcd_interrupt ( 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_Set_VoltageOn

Enables operation of internal voltage boost circuit and capacitor split circuit.

[Syntax]

```
void        R_LCD_Set_VoltageOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LCD_Set_VoltageOff

Disables operation of internal voltage boost circuit and capacitor split circuit.

[Syntax]

```
void        R_LCD_Set_VoltageOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LCD_Set_PowerOff

Halts the clock supplied to the LCD controller/driver.

Remark 1. Calling this API function changes the LCD controller/driver to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

Remark 2. This API function stops the clock supply to the LCD controller/driver, by operating the RTCEN bit of peripheral enable register *n*.
For this reason, this API function also stops the clock supply to other peripheral devices sharing the RTCEN bit (e.g. real-time clock).

[Syntax]

```
void        R_LCD_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LCD_VoltageOn

Enables operation of internal voltage boost circuit and capacitor split circuit.

[Syntax]

```
void        R_LCD_Set_VoltageOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LCD_VoltageOff

Disables operation of internal voltage boost circuit and capacitor split circuit.

[Syntax]

```
void        R_LCD_Set_VoltageOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.3.34 Sound generator

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

Table 3.36 API Functions: [Sound Generator]

API Function Name	Function
R_SG_Create	Performs initialization necessary to control the sound generator.
R_SG_Create_UserInit	Performs user-defined initialization relating to the sound generator.
r_sg_interrupt	Performs processing in response to the threshold value detection of the logarithmic decrement interrupt INTSG.
R_SG_Start	Enables operation of sound generator.
R_SG_Stop	Disables operation of sound generator.

R_SG_Create

Performs initialization necessary to control the sound generator.

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

[Syntax]

```
void        R_SG_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_SG_Create_UserInit

Performs user-defined initialization relating to the sound generator.

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

[Syntax]

```
void    R_SG_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_sg_interrupt

Performs processing in response to the threshold value detection of the logarithmic decrement interrupt INTSG.

Remark This API function is called as the interrupt process corresponding to the threshold value detection of the logarithmic decrement interrupt INTSG.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_sg_interrupt ( void );
```

CC-RL Compiler

```
static void     __near r_sg_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_SG_Start

Enables operation of sound generator.

[Syntax]

```
void        R_SG_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_SG_Stop

Disables operation of sound generator.

[Syntax]

```
void        R_SG_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.3.35 DMA controller

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

Table 3.37 API Functions: [DMA Controller]

API Function Name	Function
R_DMACn_Create	Performs initialization necessary to control the DMA controller.
R_DMACn_Create_UserInit	Performs user-defined initialization relating to the DMA controller.
R_DMAC_Create	Performs initialization necessary to control the DMA controller.
R_DMAC_Create_UserInit	Performs user-defined initialization relating to the DMA controller.
r_dmacn_interrupt	Performs processing in response to the DMA transfer end interrupt INTDMA*n*.
R_DMACn_Start	Enables operation of channel *n*.
R_DMACn_Stop	Disables operation of channel *n*.
R_DMACn_Set_SoftwareTriggerOn	Starts DMA transfer.

R_DMACn_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_DMACn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DMACn_Create_UserInit

Performs user-defined initialization relating to the DMA controller.

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

[Syntax]

```
void        R_DMACn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

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_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.

r_dmac*n*_interrupt

Performs processing in response to the DMA transfer end interrupt INTDMA*n*.

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

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_dmacn_interrupt ( void );
```

CC-RL Compiler

```
static void     __near r_dmacn_interrupt ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC*n*_Start

Enables operation of channel *n*.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC*n*_Stop

Disables operation of channel *n*.

Remark 1. This API function does not forcibly terminate DMA transfer.

Remark 2. Before using this API function, you must confirm that transmission has ended.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC*n*_Set_SoftwareTriggerOn

Starts DMA transfer.

[Syntax]

```
void        R_DMACn_Set_SoftwareTriggerOn ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

Usage example

Start data transfer by the end of A/D conversion.

(Get A/D conversion results of 4 pins and copy them to RAM. Then, calculate the average of the results.)

[GUI setting example]

DMA controller				Used
	DMA0			Used
			DMAoperation setting	Used
			Transfer direction setting	SFR to internal RAM
			Transfer data size setting	8 bits
			SFR address	ADCR - 0x000fff1e
			RAM address	0xffe00
			Transfer byte count	4
			Trigger signal	INTAD (Please set INTAD)
			DMA0 transfer end interrupt (INTDMA0)	Used
			Priority	Low

A/D convertor				Used
	ADC			Used
			A/D convertor operation setting	Used
			Comparator operation setting	Operation
			Resolution setting	8 bits
			VREF(+) setting	VDD
			VREF(-) setting	VSS
			Trigger mode setting	Software trigger mode
			Operation mode setting	Countinous select mode
			ANI0 - ANI7 analog input selection	ANI0 - ANI3
			A/D channel selection	ANI0 - ANI3
			Conversion time mode	Normal 1
			Conversion time	34 (1088/fCLK)(μs)
			Conversion result upper/lower bound value setting	Generates an interrupt request (INTAD) when ADLL≦ADCRH≦ADUL
			Upper bound (ADUL) value	255
			Lower bound (ADLL) value	0
			Use A/D interrupt (INTAD)	Used

[API setting example]

r_main.c

```
void main(void)
{
        R_MAIN_UserInit();
        /* Start user code. Do not edit comment generated here */
        /* Enable DMA0 transfer */
        R_DMAC0_Start();

        /* Start the AD converter */
        R_ADC_Start();

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

r_cg_dmac_user.c

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

/* Start user code for pragma. Do not edit comment generated here */
#pragma address (g_adc_buf = 0x0ffe00)
/* End user code. Do not edit comment generated here */

/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_adc_buf[5][4];
volatile uint8_t g_adc_buf_cnt = 0U;
/* End user code. Do not edit comment generated here */

static void __near r_dmac0_interrupt(void)
{
        /* Start user code. Do not edit comment generated here */
        uint8_t i;
        uint8_t j;
        uint16_t temp;

        /* Stop the AD converter */
        R_ADC_Stop();

        /* Disable DMA0 transfer */
        R_DMAC0_Stop();

        /* Change DMA0_RAM address */
        if ((++g_adc_buf_cnt) < 4U)
        {
                DRA0 += 4U;
        }
        else
        {
                DRA0 = _FE00_DMA0_RAM_ADDRESS;
                g_adc_buf_cnt = 0U;

                /* Calculate the average */
                for (i = 0; i < 4U; i++)
                {
                        temp = 0U;
                        for (j = 0; j < 4U; j++)
                        {
                                temp += g_adc_buf[j][i];
                        }
                        g_adc_buf[4][i] = temp / 4U;
                }
        }

        /* Enable DMA0 transfer */
        R_DMAC0_Start();

        /* Start the AD converter */
        R_ADC_Start();
        /* End user code. Do not edit comment generated here */
}
```

3.3.36 Data transfer controller

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

Table 3.38 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_DTCn_Start	Enables operation of the Data transfer controller.
R_DTCn_Stop	Disables operation of the Data transfer controller.
R_DTC_Set_PowerOff	Halts the clock supplied to the Data transfer controller.
R_DTCDn_Start	Enables operation of the Data transfer controller.
R_DTCDn_Stop	Disables operation of the Data transfer controller.

R_DTC_Create

Performs initialization necessary to control the DTC.

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 DTC.

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*n*_Start

Enables operation of the DTC.

[Syntax]

```
void        R_DTCn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DTC*n*_Stop

Disables operation of the DTC.

[Syntax]

```
void        R_DTCn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DTC_Set_PowerOff

Halts the clock supplied to the DTC.

Remark Calling this API function changes the DTC to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void        R_DTC_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DTCD*n*_Start

Enables operation of the DTC.

[Syntax]

```
void        R_DTCDn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DTCD*n*_Stop

Disables operation of the DTC.

[Syntax]

```
void        R_DTCDn_Stop ( void );
```

Remark   *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

Usage example

Start DTC data transfer by UART0 reception. (Repeat reception of 4Bytes data and copying them to the RAM array.)

[GUI setting example]

Data transfer controller				Used
	DTC			Used
		DTCBA		
			DTC base address	0xffd00
			Control data0 (DTCD0)	Used (Chain transfer:Unused; Activation sources:UART0 reception/CSI01/IIC01 transfer end or CSI01 buffer empty)
		DTCD0		
			Transfer mode setting	Repeat mode
			Repeat mode interrupt setting	Disable
			Repeat area setting	Transfer destination
			Source address	0xff12 Address fixed
			Destination address	0xfb00
			Count	4
			Block size	1

Serial				Used
	SAU0			Used
		Channel0		
			Channel 0	UART0 (Receive)
			Data length setting (Receive)	8 bits
			Trasfer direction setting (Receive)	LSB
			Parity setting (Receive)	None
			Stop bit length setting (Receive)	1 bit fixed
			Receive data level setting	Normal
			Transfer rate setting (Receive)	9600(bps) (error:+0.16% Minimum permissible value:-5.17% Maximum permissible value:+5.16%)
			Reception end interrupt priority (INTSR0)	Low
			Reception end (Callback function setting)	Used
			Reception error (Callback function setting)	Used

[API setting example]

r_main.c

```
/* Start user code for pragma. Do not edit comment generated here */
#pragma address (g_uart0_buf = 0x0ffb00)
/* End user code. Do not edit comment generated here */

/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_uart0_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 */
    /* Enable DTCD0 module operation */
    R_DTCD0_Start();

    /* Start the UART0 module operation */
    R_UART0_Start();

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

3.3.37 Event link controller

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

Table 3.39 API Functions: [Event Link Controller]

API Function Name	Function
R_ELC_Create	Performs initialization necessary to control the event link controller (ELC).
R_ELC_Create_UserInit	Performs user-defined initialization relating to the event link controller (ELC).
R_ELC_Stop	Disables operation of the event link controller (ELC).

R_ELC_Create

Performs initialization necessary to control the event link controller (ELC).

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 (ELC).

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_Stop

Disables operation of the event link controller (ELC).

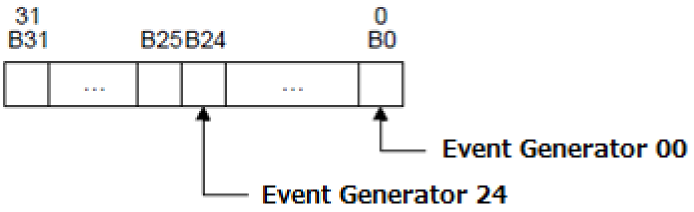
[Syntax]

```
void        R_ELC_Stop ( uint32_t event );
```

[Argument(s)]

I/O	Argument	Description
I	uint32_t *event*;	Disabled event source

Remark Below is shown the format for specifying disabled event source *event*.
In case of setting the *event* to 0x01010101, the event link operations of event source 00, 08, 16, 24 are prohibited.



[Return value]

None.

Usage example

Start A/D conversion by ‘External interrupt edge detection 0’ of ELC event. Store A/D conversion result to RAM, then stop ELC.

[GUI setting example]

Event link controller				Used
	ELC			Used
			A/D conversion starts	Used
			Event generation source	‘External interrupt edge detection 0

Interrupt				Used
	INTP			Used
		INTP0		
			Valid edge	Falling
			Priority	Low

A/D convertor				Used
	ADC			Used
			A/D convertor operation setting	Used
			Comparator operation setting	Operation
			Resolution setting	10 bits
			VREF(+) setting	VDD
			VREF(-) setting	VSS
			Trigger setting mode	Hardware trigger no wait mode
			Hardware trigger no wait mode	ELC
			Operation mode setting	One-shot select mode
			ANI0 - ANI23 analog input selection	ANI0
			A/D channel selection	ANI0
			Conversion time mode	Normal 1
			Conversion time	38 (1216/fCLK)(μs)
			Conversion result upper/lower bound value setting	Generates an interrupt request (INTAD) when ADLL≦ADCRH≦ADUL
			Upper bound (ADUL) value	255
			Lower bound (ADLL) value	0
			Use A/D interrupt (INTAD)	Used
			Priority	Low

[API setting example]

r_main.c

```
void main(void)
{
	R_MAIN_UserInit();
	/* Start user code. Do not edit comment generated here */
	/* Start the AD converter */
	R_ADC_Start();

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

r_cg_adc_user.c

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

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

static void __near r_adc_interrupt(void)
{
	/* Start user code. Do not edit comment generated here */
	/* Stop the AD converter */
	R_ADC_Stop();

	/* Return the conversion result in the buffer */
	R_ADC_Get_Result((uint16_t *)&g_adc_value);

	/* Stop the ELC event resources */
	R_ELC_Stop(0x00000001U);
	/* End user code. Do not edit comment generated here */
}
```

3.3.38 Interrupt functions

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

Table 3.40 API Functions: [Interrupt Functions]

API Function Name	Function
R_INTC_Create	Performs initialization necessary to control the interrupt functions.
R_INTC_Create_UserInit	Performs user-defined initialization relating to the interrupt functions.
r_intcn_interrupt	Performs processing in response to the external maskable interrupt INTP*n*.
R_INTCn_Start	Enables the acceptance of the external maskable interrupts INTP*n*.
R_INTCn_Stop	Disables the acceptance of the external maskable interrupts INTP*n*.
r_intclrn_interrupt	Performs processing in response to the external maskable interrupt INTPLR*n*.
R_INTCLRn_Start	Enables the acceptance of the external maskable interrupts INTPLR*n*.
R_INTCLRn_Stop	Disables the acceptance of the external maskable interrupts INTPLR*n*.
r_intrtcicn_interrupt	Performs processing in response to the external maskable interrupt INTRTCIC*n*.
R_INTRTCICn_Start	Enables the acceptance of the external maskable interrupts INTRTCIC*n*.
R_INTRTCICn_Stop	Disables the acceptance of the external maskable interrupts INTRTCIC*n*.
R_INTFO_Start	Enables the acceptance of the external maskable interrupts INTFO.
R_INTFO_Stop	Disables the acceptance of the external maskable interrupts INTFO.
R_INTFO_ClearFlag	Clears INTFCLR flag of Interrupt flag output control register 1 (INTFOCTL1).
r_intfo_interrupt	Performs processing in response to the external maskable interrupt INTFO.

R_INTC_Create

Performs initialization necessary to control the interrupt functions.

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

[Syntax]

```
void        R_INTC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_INTC_Create_UserInit

Performs user-defined initialization relating to the interrupt functions.

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

[Syntax]

```
void          R_INTC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_intc*n*_interrupt

Performs processing in response to the external maskable interrupt INTP*n*.

Remark This API function is called as the interrupt process corresponding to the external maskable interrupt INTP*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_intcn_interrupt ( void );
```

CC-RL Compiler

```
static void     __near r_intcn_interrupt ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_INTC*n*_Start

Enables the acceptance of the external maskable interrupts INTP*n*.

[Syntax]

```
void        R_INTCn_Start ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_INTC*n*_Stop

Disables the acceptance of the external maskable interrupts INTP*n*.

[Syntax]

```
void        R_INTCn_Stop ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

r_intclr*n*_interrupt

Performs processing in response to the external maskable interrupt INTPLR*n*.

Remark This API function is called as the interrupt process corresponding to the external maskable interrupt INTPLR*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_intclrn_interrupt ( void );
```

CC-RL Compiler

```
static void     __near r_intclrn_interrupt ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_INTCLR*n*_Start

Enables the acceptance of the external maskable interrupts INTPLR*n*.

[Syntax]

```
void        R_INTCLRn_Start ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_INTCLR*n*_Stop

Disables the acceptance of the external maskable interrupts INTPLR*n*.

[Syntax]

```
void        R_INTCLRn_Stop ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

r_intrtcic*n*_interrupt

Performs processing in response to the external maskable interrupt INTRTCIC*n*.

Remark This API function is called as the interrupt process corresponding to the external maskable interrupt INTRTCIC*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_intrtcicn_interrupt ( void );
```

CC-RL Compiler

```
static void     __near r_intrtcicn_interrupt ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_INTRTCIC*n*_Start

Enables the acceptance of the external maskable interrupts INTRTCIC*n*.

[Syntax]

```
void        R_INTRTCICn_Start ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_INTRTCIC*n*_Stop

Disables the acceptance of the external maskable interrupts INTRTCIC*n*.

[Syntax]

```
void        R_INTRTCICn_Stop ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_INTFO_Start

Enables the acceptance of the external maskable interrupts INTFO.

[Syntax]

```
void        R_INTFO_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_INTFO_Stop

Disables the acceptance of the external maskable interrupts INTFO.

[Syntax]

```
void        R_INTFO_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_INTFO_ClearFlag

Clears INTFCLR flag of Interrupt flag output control register 1 (INTFOCTL1).

[Syntax]

```
void        R_INTFO_ClearFlag ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_intfo_interrupt

Performs processing in response to the external maskable interrupt INTFO.

Remark This API function is called as the interrupt process corresponding to the external maskable interrupt INTFO.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_intfo_interrupt ( void );
```

CC-RL Compiler

```
static void     __near r_intfo_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

Usage example

Count the number of falling edge.

[GUI setting example]

Interrupt				Used
	INTP			Used
		INTP0		
			Valid edge	Falling
			Priority	Low

[API setting example]

r_main.c

```
void main(void)
{
        R_MAIN_UserInit();
        /* Start user code. Do not edit comment generated here */
        /* Clear INTP0 interrupt flag and enable interrupt */
        R_INTC0_Start();

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

r_cg_intc_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_intc0_cnt = 0U;
/* End user code. Do not edit comment generated here */

static void __near r_intc0_interrupt(void)
{
        /* Start user code. Do not edit comment generated here */
        /* Count INTP0 */
        g_intc0_cnt++;
        /* End user code. Do not edit comment generated here */
}
```

3.3.39 Key interrupt function

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

Table 3.41 API Functions: [Key Interrupt Function]

API Function Name	Function
R_KEY_Create	Performs initialization necessary to control the key interrupt function.
R_KEY_Create_UserInit	Performs user-defined initialization relating to the key interrupt function.
r_key_interrupt	Performs processing in response to the key interrupt INTKR.
R_KEY_Start	Enables the acceptance of the key interrupt INTKR.
R_KEY_Stop	Disables the acceptance of the key interrupt INTKR.

R_KEY_Create

Performs initialization necessary to control the key interrupt function.

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

[Syntax]

```
void        R_KEY_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_KEY_Create_UserInit

Performs user-defined initialization relating to the key interrupt function.

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

[Syntax]

```
void R_KEY_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_key_interrupt

Performs processing in response to the key interrupt INTKR.

Remark This API function is called as the interrupt process corresponding to the key interrupt INTKR.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_key_interrupt ( void );
```

CC-RL Compiler

```
static void     __near r_key_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_KEY_Start

Enables the acceptance of the key interrupt INTKR.

[Syntax]

```
void        R_KEY_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_KEY_Stop

Disables the acceptance of the key interrupt INTKR.

[Syntax]

```
void        R_KEY_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

Usage example

Set the flag for 8 keys which the falling edge is detected.

[GUI setting example]

Interrupt				Used
	KEY			Used
			KR0	Used
			KR1	Used
			KR2	Used
			KR3	Used
			KR4	Used
			KR5	Used
			KR6	Used
			KR7	Used
			Priority	High

[API setting example]

r_main.c

```
void main(void)
{
	R_MAIN_UserInit();
	/* Start user code. Do not edit comment generated here */
	/* Clear INTKR interrupt flag and enable interrupt */
	R_KEY_Start();

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

r_cg_intc_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_key_fix = 0x00U;
/* End user code. Do not edit comment generated here */

static void __near r_key_interrupt(void)
{
	/* Start user code. Do not edit comment generated here */
	g_key_fix = ~P7;
	/* End user code. Do not edit comment generated here */
}
```

3.3.40 Voltage detector

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

Table 3.42 API Functions: [Voltage Detector]

API Function Name	Function
R_LVD_Create	Performs initialization necessary to control the voltage detector.
R_LVD_Create_UserInit	Performs user-defined initialization relating to the voltage detector.
r_lvd_interrupt	Performs processing in response to the voltage detection interrupt INTLVI.
r_lvd_vddinterrupt	Performs processing in response to the voltage detection of VDD pin interrupt INTLVDVDD.
r_lvd_vbatinterrupt	Performs processing in response to the voltage detection of VBAT pin interrupt INTLVDVBAT.
r_lvd_vrtcinterrupt	Performs processing in response to the voltage detection of VRTC pin interrupt INTLVDVRTC.
r_lvd_exlvdinterrupt	Performs processing in response to the voltage detection of EXLVD pin interrupt INTLVDEXLVD.
R_LVD_InterruptMode_Start	Starts voltage detection (when in interrupt mode, and interrupt & reset mode).
R_LVD_Start_VDD	Enables operation of VDD pin voltage detection.
R_LVD_Start_VBAT	Enables operation of VBAT pin voltage detection.
R_LVD_Start_VRTC	Enables operation of VRTC pin voltage detection.
R_LVD_Start_EXLVD	Enables operation of EXLVD pin voltage detection.
R_LVD_Stop_VDD	Disables operation of VDD pin voltage detection.
R_LVD_Stop_VBAT	Disables operation of VBAT pin voltage detection.
R_LVD_Stop_VRTC	Disables operation of VRTC pin voltage detection.
R_LVD_Stop_EXLVD	Disables operation of EXLVD pin voltage detection.
R_LVI_Create	Performs initialization necessary to control the voltage detector.
R_LVI_Create_UserInit	Performs user-defined initialization relating to the voltage detector.
r_lvi_interrupt	Performs processing in response to the voltage detection interrupt INTLVI.
R_LVI_InterruptMode_Start	Starts voltage detection (when in interrupt mode, and interrupt & reset mode).

R_LVD_Create

Performs initialization necessary to control the voltage detector.

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

[Syntax]

```
void        R_LVD_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVD_Create_UserInit

Performs user-defined initialization relating to the voltage detector.

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

[Syntax]

```
void        R_LVD_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_lvd_interrupt

Performs processing in response to the voltage detection interrupt INTLVI.

Remark This API function is called as the interrupt process corresponding to the voltage detection interrupt INTLVI.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_lvd_interrupt ( void );
```

CC-RL Compiler

```
static void    __near r_lvd_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_lvd_vddinterrupt

Performs processing in response to the voltage detection of VDD pin interrupt INTLVDVDD.

Remark This API function is called as the interrupt process corresponding to the voltage detection of VDD pin interrupt INTLVDVDD.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_lvd_vddinterrupt ( void );
```

CC-RL Compiler

```
static void     __near r_lvd_vddinterrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_lvd_vbatinterrupt

Performs processing in response to the voltage detection of VBAT pin interrupt INTLVDVBAT.

Remark This API function is called as the interrupt process corresponding to the voltage detection of VBAT pin interrupt INTLVDVBAT.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_lvd_vbatinterrupt ( void );
```

CC-RL Compiler

```
static void     __near r_lvd_vbatinterrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_lvd_vrtcinterrupt

Performs processing in response to the voltage detection of VRTC pin interrupt INTLVDVRTC.

Remark This API function is called as the interrupt process corresponding to the voltage detection of VRTC pin interrupt INTLVDVRTC.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_lvd_vrtcinterrupt ( void );
```

CC-RL Compiler

```
static void     __near r_lvd_vrtcinterrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_lvd_exlvdinterrupt

Performs processing in response to the voltage detection of EXLVD pin interrupt INTLVDEXLVD.

Remark This API function is called as the interrupt process corresponding to the voltage detection of EXLVD pin interrupt INTLVDEXLVD.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_lvd_exlvdinterrupt ( void );
```

CC-RL Compiler

```
static void     __near r_lvd_exlvdinterrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVD_InterruptMode_Start

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

[Syntax]

```
void        R_LVD_InterruptMode_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVD_Start_VDD

Enables operation of VDD pin voltage detection.

[Syntax]

```
void        R_LVD_Start_VDD ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVD_Start_VBAT

Enables operation of VBAT pin voltage detection.

[Syntax]

```
void        R_LVD_Start_VBAT ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVD_Start_VRTC

Enables operation of VRTC pin voltage detection.

[Syntax]

```
void        R_LVD_Start_VRTC ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVD_Start_EXLVD

Enables operation of EXLVD pin voltage detection.

[Syntax]

```
void        R_LVD_Start_EXLVD ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVD_Stop_VDD

Disables operation of VDD pin voltage detection.

[Syntax]

```
void        R_LVD_Stop_VDD ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVD_Stop_VBAT

Disables operation of VBAT pin voltage detection.

[Syntax]

```
void        R_LVD_Stop_VBAT ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVD_Stop_VRTC

Disables operation of VRTC pin voltage detection.

[Syntax]

```
void        R_LVD_Stop_VRTC ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVD_Stop_EXLVD

Disables operation of EXLVD pin voltage detection.

[Syntax]

```
void        R_LVD_Stop_EXLVD ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVI_Create

Performs initialization necessary to control the voltage detector.

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

[Syntax]

```
void        R_LVI_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVI_Create_UserInit

Performs user-defined initialization relating to the voltage detector.

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

[Syntax]

```
void R_LVI_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_lvi_interrupt

Performs processing in response to the voltage detection interrupt INTLVI.

Remark This API function is called as the interrupt process corresponding to the voltage detection interrupt INTLVI.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_lvi_interrupt ( void );
```

CC-RL Compiler

```
static void     __near r_lvi_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_LVI_InterruptMode_Start

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

[Syntax]

```
void        R_LVI_InterruptMode_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

Usage example

Detect the falling of the supply voltage by Interrupt & reset mode, and perform arbitrary processing in interrupt function.

[GUI setting example]

Voltage detector				
	LVD			Used
			Low voltage detector operation setting	Used
			Operation mode setting	Interrupt & reset mode
			INTLVI Priority	Low
			Reset generation level (VLVDL)	2.75(V)
			Interrupt generation level (VLVDH)	3.15(V)

[API setting example]

r_main.c

```
void main(void)
{
        R_MAIN_UserInit();
        /* Start user code. Do not edit comment generated here */
        /* Enable the voltage detector interrupt */
        R_LVD_InterruptMode_Start();

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

r_cg_lvd_user.c

```
static void __near r_lvd_interrupt(void)
{
        /* Start user code. Do not edit comment generated here */
        /*** Processing to be performed before the reset ***/
        /* End user code. Do not edit comment generated here */
}
```

3.3.41 Battery backup function

Below is a list of API functions output by the Code Generator for battery backup function use.

Table 3.43 API Functions: [Battery Backup Function]

API Function Name	Function
R_BUP_Create	Performs initialization necessary to control the battery backup function.
R_BUP_Create_UserInit	Performs user-defined initialization relating to the battery backup function.
r_bup_interrupt	Performs processing in response to the power switching detection interrupt INTVBAT.
R_BUP_Start	Enables operation of battery backup function.
R_BUP_Stop	Disables operation of battery backup function.

R_BUP_Create

Performs initialization necessary to control the battery backup function.

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

[Syntax]

```
void R_BUP_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_BUP_Create_UserInit

Performs user-defined initialization relating to the battery backup function.

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

[Syntax]

```
void        R_BUP_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_bup_interrupt

Performs processing in response to the power switching detection interrupt INTVBAT.

Remark This API function is called as the interrupt process corresponding to the power switching detection interrupt INTVBAT.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_bup_interrupt ( void );
```

CC-RL Compiler

```
static void    __near r_bup_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_BUP_Start

Enables operation of battery backup function.

[Syntax]

```
void        R_BUP_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_BUP_Stop

Disables operation of battery backup function.

[Syntax]

```
void        R_BUP_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

Usage example

Set the flag to notify switching to battery backup mode.

[GUI setting example]

Battery backup function				Used
	KEYBATTERYBACKUP			Used
			Power switching operation setting	Used
			Generate an interrupt when power is switched (INTVBAT)	Used
			Power switching interrupt selection	Interrupt generated when VDD is switched to VBAT
			Priority	Low

[API setting example]

r_cg_main.c

```
void main(void)
{
        R_MAIN_UserInit();
        /* Start user code. Do not edit comment generated here */
        /* Start battery backup module operation */
        R_BUP_Start();

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

r_cg_bup_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_bup_f = 0U;
/* End user code. Do not edit comment generated here */

static void __near r_bup_interrupt(void)
{
        /* Start user code. Do not edit comment generated here */
        /* When entering the battery backup mode, a flag is set */
        if (VBATCMPM == 0U) {
                g_bup_f = 0U;
        }
        else
        {
                g_bup_f = 1U;
        }
        /* End user code. Do not edit comment generated here */
}
```

3.3.42 Oscillation stop detector

Below is a list of API functions output by the Code Generator for oscillation stop detector use.

Table 3.44 API Functions: [Oscillation Stop Detector]

API Function Name	Function
R_OSDC_Create	Performs initialization necessary to control the oscillation stop detector.
R_OSDC_Create_UserInit	Performs user-defined initialization relating to the oscillation stop detector.
r_osdc_interrupt	Performs processing in response to the oscillation stop detection interrupt INTOSDC.
R_OSDC_Start	Enables operation of oscillation stop detector.
R_OSDC_Stop	Disables operation of oscillation stop detector.
R_OSDC_Set_PowerOff	Halts the clock supplied to the oscillation stop detector.
R_OSDC_Reset	Reset the oscillation stop detector.

R_OSDC_Create

Performs initialization necessary to control the oscillation stop detector.

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

[Syntax]

```
void        R_OSDC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_OSDC_Create_UserInit

Performs user-defined initialization relating to the oscillation stop detector.

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

[Syntax]

```
void        R_OSDC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_osdc_interrupt

Performs processing in response to the oscillation stop detection interrupt INTOSDC.

Remark This API function is called as the interrupt process corresponding to the oscillation stop detection interrupt INTOSDC.

[Syntax]

CA78K0R Compiler

```
__interrupt static void       r_osdc_interrupt ( void );
```

CC-RL Compiler

```
static void       __near r_osdc_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_OSDC_Start

Enables operation of oscillation stop detector.

[Syntax]

```
void        R_OSDC_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_OSDC_Stop

Disables operation of oscillation stop detector.

[Syntax]

```
void          R_OSDC_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_OSDC_Set_PowerOff

Halts the clock supplied to the oscillation stop detector.

Remark Calling this API function changes the oscillation stop detector to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void        R_OSDC_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_OSDC_Reset

Reset the oscillation stop detector.

[Syntax]

```
void        R_OSDC_Reset ( void );
```

[Argument(s)]

None.

[Return value]

None.

Usage example

Reset by Oscillation stop detection interrup.

[GUI setting example]

Oscillation stop detector				Used
	OSCSTOPDETECTOR			Used
			Oscillation stop detector operation setting	Used
			Oscillation stop judgement time	100(ms) (TYP.) (Actual value：100)
			Generate an interrupt when oscillation stop is detected (INTOSDC)	Used
			Priority	Low

Watchdog timer				
	WDT			Used
			Watchdog timer operationsetting	Used
			Operation in HALT/STOP/SNOOZE mode setting	Enaled
			Overflow time	2^16/fIL 4369.07(ms)
			Window open period	100(%)
			Enable interval interruptwhen 75% + 1/2fIL of overflow time (INTWDTI)	Used
			Priority	Low

[API setting example]

r_main.c

```
void main(void)
{
	R_MAIN_UserInit();
	/* Start user code. Do not edit comment generated here */
	/* Start OSDC module operation */
	R_OSDC_Start();

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

r_cg_osdc_user.c

```
static void __near r_osdc_interrupt(void)
{
	/* Start user code. Do not edit comment generated here */
	/* Stop OSDC module operation */
	R_OSDC_Stop();

	/* End user code. Do not edit comment generated here */
}
```

3.3.43 SPI interface

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

Table 3.45 API Functions: [SPI Interface]

API Function Name	Function
R_SAIC_Create	Performs initialization necessary to control the SPI interface.
R_SAIC_Create_UserInit	Performs user-defined initialization relating to the SPI interface.
R_SAIC_Write	Starts SPI data transmission.
R_SAIC_Read	Starts SPI data reception.
R_SPI_Create	Performs initialization necessary to control the SPI interface.
R_SPI_Create_UserInit	Performs user-defined initialization relating to the SPI interface.
R_SPI_Write	Starts SPI data transmission.
R_SPI_Read	Starts SPI data reception.

R_SAIC_Create

Performs initialization necessary to control the SPI interface.

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

[Syntax]

```
void        R_SAIC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_SAIC_Create_UserInit

Performs user-defined initialization relating to the SPI interface.

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

[Syntax]

```
void R_SAIC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_SAIC_Write

Starts SPI data transmission.

[Syntax]

```
void        R_SAIC_Write ( const smartanalog_t * p_saic_data );
```

[Argument(s)]

I/O	Argument	Description
I	const smartanalog_t * *p_saic_data*;	Pointer to area storing the transmission data

[Return value]

None.

R_SAIC_Read

Starts SPI data reception.

[Syntax]

```
void        R_SAIC_Read ( const smartanalog_t * p_saic_data, smartanalog_t * p_saic_read_buf );
```

[Argument(s)]

I/O	Argument	Description
O	const smartanalog_t * *p_saic_data*;	Pointer to area to store the received data
O	smartanalog_t * *p_saic_read_buf*;	Pointer to a buffer to store the received data

[Return value]

None.

R_SPI_Create

Performs initialization necessary to control the SPI interface.

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

[Syntax]

```
void        R_SPI_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_SPI_Create_UserInit

Performs user-defined initialization relating to the SPI interface.

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

[Syntax]

```
void        R_SPI_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_SPI_Write

Starts SPI data transmission.

[Syntax]

```
void        R_SPI_Write ( const smartanalog_t * p_saic_data );
```

[Argument(s)]

I/O	Argument	Description
I	const smartanalog_t * *p_saic_data*;	Pointer to area storing the transmission data

[Return value]

None.

R_SPI_Read

Starts SPI data reception.

[Syntax]

```
void        R_SPI_Read ( const smartanalog_t * p_saic_data, smartanalog_t * p_saic_read_buf );
```

[Argument(s)]

I/O	Argument	Description
O	const smartanalog_t * *p_saic_data*;	Pointer to area to store the received data
O	smartanalog_t * *p_saic_read_buf*;	Pointer to a buffer to store the received data

[Return value]

None.

Usage example

Transfer and write data ot external device by SPI transmission. After that, read the data for verify check.
※Please use SA-Designer together.

[GUI setting example]

SPI interface				
	SPI			Used
			Analog IC operation setting	Used
			Baudrate	300(bps) (Actual value：300.481)

[API setting example]

r_cg_main.c

```
/* Start user code for global. Do not edit comment generated here */
uint8_t g_flag;
smartanalog_t gp_sa_read_buf[];
extern const smartanalog_t gp_smartanalog_data[];
/* 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)
        {
                if (1U == g_flag)
                {
                        /* Write SAIC register */
                        R_SAIC_Write(gp_smartanalog_data);

                        /* Read SAIC register */
                        R_SAIC_Read(gp_smartanalog_data, gp_sa_read_buf);

                        /** read after write verify **/
                }
        }
        /* End user code. Do not edit comment generated here */
}
```

3.3.44 Operational amplifier

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

Table 3.46 API Functions: [Operational amplifier]

API Function Name	Function
R_OPAMP_Create	Performs initialization necessary to control the operational amplifier.
R_OPAMP_Create_UserInit	Performs user-defined initialization relatede to the operational amplifier.
R_OPAMP_Set_ReferenceCircuitOn	Enables operational amplifier reference current circuit.
R_OPAMP_Set_ReferenceCircuitOff	Disables operational amplifier reference current circuit.
R_OPAMPn_Start	Starts operational amplifier of unit *n*.
R_OPAMPn_Stop	Stops operational amplifier of unit *n*.
R_OPAMPn_Set_PrechargeOn	Starts precharging of the external capacitor of the operational amplifier *n*.
R_OPAMPn_Set_PrechargeOff	Performs user-defined initialization relatede to the operational amplifier *n*.

R_OPAMP_Create

Performs initialization necessary to control the operational amplifier.

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

[Syntax]

```
void R_OPAMP_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_OPAMP_Create_UserInit

Performs user-defined initialization relating to the operational amplifier.

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

[Syntax]

```
void        R_OPAMP_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_OPAMP_Set_ReferenceCircuitOn

Enables operational amplifier reference current circuit.

[Syntax]

```
void        R_OPAMP_Set_ReferenceCircuitOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_OPAMP_Set_ReferenceCircuitOff

Disables operational amplifier reference current circuit.

[Syntax]

```
void        R_OPAMP_Set_ReferenceCircuitOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_OPAMP*n*_Start

Starts operational amplifier of unit *n*.

[Syntax]

```
void        R_OPAMPn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_OPAMP*n*_Stop

Stops operational amplifier of unit *n*.

[Syntax]

```
void        R_OPAMPn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_OPAMP*n*_Set_PrechargeOn

Starts precharging of the external capacitor of the operational amplifier *n*.

[Syntax]

```
void        R_OPAMPn_Set_PrechargeOn ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_OPAMP*n*_Set_PrechargeOff

Stops precharging of the external capacitor of the operational amplifier *n*.

[Syntax]

```
void        R_OPAMPn_Set_PrechargeOff ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

Usage example

Use Operational amplifer for Comparator as + side input.

[GUI setting example]

Operational amplifer				
	OPAMP			Used
			Operational amplifer operation setting	Used
			Use operational amplifer 0	Used
			Use operational amplifer 1	Unused
			Use operational amplifer 2	Unused
			Use operational amplifer 3	Unused
			Reference current circuit setting	Stop
			Operation mode setting	Low-power mode
			ELC trigger setting	Operational amplifer 0: Opeartional amplifer ELC trigger 0
				Operational amplifer 1: Opeartional amplifer ELC trigger 1
				Operational amplifer 2: Opeartional amplifer ELC trigger 2
				Operational amplifer 3: Opeartional amplifer ELC trigger 3
		Operational Amplifier 0		Used
			Activation/stop trigger control setting	Software trigger mode

Comparator				
	COMP			Used
			Operation setting	Use only comparator 0 (Comparator input:AMP0O)
			Speed setting	Low speed
		Comparator 0		Used
			Comparator0 Mode setting	Normal
			Comparator0 Edge setting	Rising edge
			Comparator0 Digital filter setting	Unused
			Comparator0 Enable output (VCOUT0)	Unused
			Comparator0 Use comparator 0 interrupt (INTCMP0)	Used
			Comparator0 Priority	Low

[API setting example]

r_cg_main.c

```
void main(void)
{
	R_MAIN_UserInit();
	/* Start user code. Do not edit comment generated here */
	/* Start the operational amplifier 0 */
	R_OPAMP0_Start();

	/* Start the comparator 0 */
	R_COMP0_Start();

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

3.3.45 Data operation circuit

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

Table 3.47 API Functions: [Operational amplifier]

API Function Name	Function
R_DOC_Create	Performs initialization necessary to control the data operation circuit.
R_DOC_Create_UserInit	Performs user-defined initialization related to the data operation circuit.
r_doc_interrupt	Performs processing in response to the DOC operation result detection interrupt INTDOC.
R_DOC_SetMode	Configures the operation mode of data operation circuit.
R_DOC_WriteData	Writes new data to compare, add or subtract.
R_DOC_GetResult	Gets result of addition or subtraction.
R_DOC_ClearFlag	Clears DOPCF flag of DOC control register (DOCR).
R_DOC_Set_PowerOff	Stops the clock supplied for data operation circuit.
R_DOC_Reset	Resets Data opration circuit module.

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_interrupt

Performs processing in response to the data operation circuit interrupt INTDOC.

Remark This API function is called as the interrupt process corresponding to the data operation circuit interrupt INTDOC.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_doc_interrupt ( void );
```

CC-RL Compiler

```
static void     __near r_doc_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DOC_SetMode

Configures the operation mode of data operation circuit.

[Syntax]

```
#include        "r_cg_macrodriver.h"
#include        "r_cg_doc.h"
MD_STATUS   R_DOC_SetMode ( doc_mode_t mode, unit16_t value );
```

[Argument(s)]

I/O	Argument	Description
I	doc_mode_t *mode*;	Operation mode of data operation circuit ADDITION : Data addition mode SUBTRACTION : Data subtraction mode COMPARE_MATCH : Data comparison mode (Detection Condition: Data match is detected) COMPARE_MISMATCH : Data subtraction mode (Detection Condition: Data mismatch is detected)
I	uint16_t *value*;	Data addition and data subtraction : Results of operations Data comparison : 16-bit data for use as a reference

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_DOC_WriteData

Writes new data to compare, add or subtract.

Remark Write data to DODIR register.

[Syntax]

```
#include "r_cg_macrodriver.h"
void R_DOC_WriteData ( unit16_t data );
```

[Argument(s)]

I/O	Argument	Description
I	unit16_t *data*;	data to compare, add or subtract

[Return value]

None.

R_DOC_GetResult

Gets result of addition or subtraction.

[Syntax]

```
#include      "r_cg_macrodriver.h"
void          R_DOC_GetResult ( unit16_t*const data );
```

[Argument(s)]

I/O	Argument	Description
O	unit16_t*const *data*;	pointer to where result will be stored

[Return value]

None.

R_DOC_ClearFlag

Clears DOPCF flag of DOC control register (DOCR).

[Syntax]

```
void        R_DOC_ClearFlag ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DOC_Set_PowerOff

Stops the clock supplied for data operation circuit.

[Syntax]

```
void          R_DOC_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_DOC_Reset

Resets Data operation circuit module.

[Syntax]

```
void        R_DOC_Reset ( void );
```

[Argument(s)]

None.

[Return value]

None.

Usage example

Add array data by Data addition mode and get the result when the result is larger than “FFFFh”.

After that, change the mode to Data comparision mode and generate interrupt when detecting the data other than “0000h”.

[GUI setting example]

Data operation circuit				
	DOC			Used
			DOC setting	Used
			Data operation setting	Data addition mode
			COmparision reference/Initial value of addition or subtraction result	0xFFFF
			Enable data operation circuit interrupt (INTDOC)	Used
			INTDOC Priority	Low

[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 */

static void __near r_doc_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 DOPCF flag */
    R_DOC_ClearFlag();
    /* End user code. Do not edit comment generated here */
}
```

3.3.46 32-bit Multiply-accumulator

Below is a list of API functions output by the Code Generator for 32-bit Multiply-accumulator use.

Table 3.48 API Functions: [32-bit Multiply-accumulator]

API Function Name	Function
R_MAC32Bit_Create	Performs initialization necessary to control the 32-bit Multiply-accumulator.
R_MAC32Bit_Create_UserInit	Performs user-defined initialization relating to the 32-bit Multiply-accumulator.
r_mac32bit_interrupt_flow	Performs processing in response to the 32-bit Multiply-accumulator interrupt INTMACLOF.
R_MAC32Bit_Reset	Resets the 32-bit Multiply-accumulator.
R_MAC32Bit_Set_PowerOff	Stops the clock supplied for the 32-bit Multiply-accumulator.
R_MAC32Bit_MULUnsigned	Operates the unsigned multiply.
R_MAC32Bit_MULSigned	Operates the signed multiply.
R_MAC32Bit_MACUnsigned	Operates the unsigned multiply-accumulate.
R_MAC32Bit_MACSigned	Operates the signed multiply-accumulate.

R_MAC32Bit_Create

Performs initialization necessary to control the 32-bit Multiply-accumulator.

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

[Syntax]

```
void        R_MAC32Bit_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_MAC32Bit_Create_UserInit

Performs user-defined initialization relating to the 32-bit Multiply-accumulator.

Remark This API function is called as the R_MAC32Bit_Create callback routine.

[Syntax]

```
void        R_MAC32Bit_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_mac32bit_interrupt_flow

Performs processing in response to the 32-bit Multiply-accumulator interrupt INTMACLOF.

Remark This API function is called as the interrupt process corresponding to the 32-bit Multiply-accumulator interrupt INTMACLOF.

[Syntax]

CA78K0R Compiler

```
__interrupt static void	r_mac32bit_interrupt_flow ( void );
```

CC-RL Compiler

```
static void	__near r_mac32bit_interrupt_flow ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_MAC32Bit_Reset

Resets the 32-bit Multiply-accumulator.

[Syntax]

```
void        R_MAC32Bit_Reset ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_MAC32Bit_Set_PowerOff

Stops the clock supplied for the 32-bit Multiply-accumulator.

Remark 32-bit Multiply-accumulator enters the reset state by calling this API.Therefore, writing to the registers which control 32-bit Multiply-accumulator is ignored after calling this API.

[Syntax]

```
void        R_MAC32Bit_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_MAC32Bit_MULUnsigned

Operates the unsigned multiply.

[Syntax]

```
#include        "r_cg_macrodriver.h"
#include        "r_cg_mac32bit.h"
void        R_MAC32Bit_MULUnsigned ( uint32_t data_a, uint32_t data_b, mac32bit_uint64_t
* buffer_64bit );
```

[Argument(s)]

I/O	Argument	Description
I	uint32_t *data_a*;	Multiplicand
I	uint32_t *data_b*;	Multiplier
O	mac32bit_uint64_t * *buffer_64bit*;	Multiplier result

Remark Below is an example of the structure mac32bit_uint64_t for the multiplier result.

```
typedef struct
{
        uint16_t low_low;
        uint16_t low_high;
        uint16_t high_low;
        uint16_t high_high;
} mac32bit_uint64_t;
```

[Return value]

None.

R_MAC32Bit_MULSigned

Operates the signed multiply.

[Syntax]

```
#include        "r_cg_macrodriver.h"
#include        "r_cg_mac32bit.h"
void        R_MAC32Bit_MULSigned ( int32_t data_a, int32_t data_b, mac32bit_int64_t
* buffer_64bit );
```

[Argument(s)]

I/O	Argument	Description
I	int32_t *data_a*;	Multiplicand
I	int32_t *data_b*;	Multiplier
O	mac32bit_int64_t * *buffer_64bit*;	Multiplier result

Remark Below is an example of the structure mac32bit_int64_t for the multiplier result.

```
typedef struct
{
                int16_t low_low;
                int16_t low_high;
                int16_t high_low;
                int16_t high_high;
} mac32bit_int64_t;
```

[Return value]

None.

R_MAC32Bit_MACUnsigned

Operates the unsigned multiply-accumulate.

[Syntax]

#include "r_cg_macrodriver.h"
#include "r_cg_mac32bit.h"
void R_MAC32Bit_MACUnsigned (uint32_t *data_a*, uint32_t *data_b*, mac32bit_uint64_t * *buffer_64bit*);

[Argument(s)]

I/O	Argument	Description
I	uint32_t *data_a*;	Multiplicand
I	uint32_t *data_b*;	Multiplier
O	mac32bit_uint64_t * *buffer_64bit*;	Accumulation initial value ／ Multiplier result

Remark See R_MAC32Bit_MULUnsigned for details about the mac32bit_uint64_t.

[Return value]

None.

R_MAC32Bit_MACSigned

Operates the signed multiply-accumulate.

[Syntax]

```
#include        "r_cg_macrodriver.h"
#include        "r_cg_mac32bit.h"
void        R_MAC32Bit_MACSigned ( int32_t data_a, int32_t data_b, mac32bit_int64_t
* buffer_64bit );
```

[Argument(s)]

I/O	Argument	Description
I	int32_t *data_a*;	Multiplicand
I	int32_t *data_b*;	Multiplier
O	mac32bit_int64_t * *buffer_64bit*;	Accumulation initial value ／ Multiplier result

Remark   See R_MAC32Bit_MULSigned for details about the mac32bit_int64_t.

[Return value]

None.

Usage example

Get A/D conversion results of 1 pin 4 times. Then, calculate the average of the results.

[GUI setting example]

32-bit multiply-accumlator				
	MAC32bit			Used
			32-bit multiply-accumlator operation setting	Used
			Fixed point mode setting	Disable
			Enable multiply-accumulation overflow/underflow (INTMACLOF）	Unused

A/Dconvertor				
	ADC			Used
			A/D convertor operation setting	Used
			Comparator operation setting	Stop
			Resolution	10 bits
			VREF(+) setting	VDD
			VREF(-) setting	VSS
			Trigger setting mode	Software trigger mode
			Operation mode setting	One-shot select mode
			ANI0 – ANI5 analog input selection	ANI0
			A/D channel selection	ANI0
			Conversion time mode	Normal 1
			Conversion time	608/fCLK 25.3333(μs)
			Conversion result upper/lower bound value setting	Generates an interrupt request (INTAD) when ADLL ≦ADCRH≦ADUL
			Upper bound (ADUL) value	255
			Lower bound (ADLL) value	0
			Use A/D interrupt (INTAD)	Unused

[API setting example]

r_cg_main.c

```
/* Start user code for global. Do not edit comment generated here */
volatile mac32bit_uint64_t g_mac32bit_buf;
volatile uint16_t g_adc_fix;
volatile uint16_t g_buffer;
volatile uint8_t g_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 */
    /* Start the AD converter */
    R_ADC_Start();

    while (1U)
    {
        while (0U == ADIF)
        {
            ;
        }
        ADIF = 0U;

        /* Return the conversion result in the buffer */
        R_ADC_Get_Result((uint16_t *)&g_buffer);

        /* Caculate unsigned values in multiply-accumulation mode */
        R_MAC32Bit_MACUnsigned(1U, g_buffer, (mac32bit_uint64_t *)&g_mac32bit_buf);

        if ((++g_cnt) >= 4U)
        {
            g_cnt = 0U;
            g_adc_fix = (g_mac32bit_buf.low_low >> 4U);
            g_mac32bit_buf.low_low    = 0U;
            g_mac32bit_buf.low_high   = 0U;
            g_mac32bit_buf.high_low   = 0U;
            g_mac32bit_buf.high_high = 0U;
        }
    }
    /* End user code. Do not edit comment generated here */
}
```

3.3.47 12-bit A/D converter

Below is a list of API functions output by the Code Generator for 12-bit A/D converter use.

Table 3.49 API Functions: [12-bit A/D Converter]

API Function Name	Function
R_12ADC_Create	Performs initialization necessary to control the 12-bit A/D converter.
R_12ADC_Create_UserInit	Performs user-defined initialization relating to the 12-bit A/D converter.
r_12adc_interrupt	Performs processing in response to the A/D conversion end interrupt INTAD.
R_12ADC_Start	Starts A/D conversion.
R_12ADC_Stop	Ends A/D conversion.
R_12ADC_Get_ValueResult	Reads the results of A/D conversion (12 bits).
R_12ADC_Set_ADChannel	Configures the analog voltage input pin for A/D conversion.
R_12ADC_TemperatureSensorOutput_On	Enables 12-bit A/D converter temperature sensor output circuit.
R_12ADC_TemperatureSensorOutput_Off	Disables12-bit A/D converter temperature sensor output circuit.
R_12ADC_InternalReferenceVoltage_On	Enables 12-bit A/D converter reference voltage circuit.
R_12ADC_InternalReferenceVoltage_Off	Disables12-bit A/D converter reference voltage circuit.
R_12ADC_Set_PowerOff	Halts the clock supplied to the 12-bit A/D converter.

R_12ADC_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_12ADC_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_12ADC_Create_UserInit

Performs user-defined initialization relating to the 12-bit A/D converter.

Remark This API function is called as the R_12ADC_Create callback routine.

[Syntax]

```
void        R_12ADC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_12adc_interrupt

Performs processing in response to the A/D conversion end interrupt INTAD.

Remark This API function is called as the interrupt process corresponding to the A/D conversion end interrupt INTAD.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_12adc_interrupt ( void );
```

CC-RL Compiler

```
static void    __near r_12adc_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_12ADC_Start

Starts A/D conversion.

Remark About 1 micro second of stabilization time is required when changing the voltage converter from operation stopped to operation enabled status.
Consequently, about 1 micro second must be left free between the call to R_12ADC_Create and the call to this API function.

[Syntax]

```
void        R_12ADC_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_12ADC_Stop

Ends A/D conversion.

Remark The voltage converter continues to operate after the process of this API function completes.
Consequently, to stop the operation of the voltage converter, you must call R_12ADC_Set_PowerOff after the process of this API function completes.

[Syntax]

```
void        R_12ADC_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_12ADC_Get_ValueResult

Reads the results of A/D conversion (12 bits).

[Syntax]

```
#include      "r_cg_macrodriver.h"
MD_STATUS     R_12ADC_Get_ValueResult ( ad_channel_t channel, uint16_t * const buffer );
```

[Argument(s)]

I/O	Argument	Description
I	ad_channel_t *channel*;	Analog voltage input pin
O	uint16_t * const *buffer*;	Pointer to area in which to store read results of A/D conversion

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_12ADC_Set_ADChannel

Configures the analog voltage input pin for A/D conversion.

Remark The value specified in argument channel is set to A/D channel select register A0 (ADANSA0) or A/D conversion extended input control register (ADEXICR).

[Syntax]

```
#include        "r_cg_macrodriver.h"
#include        "r_cg_12adc.h"
MD_STATUS       R_12ADC_Set_ADChannel ( ad_sel_regester_t regester, uint16_t data );
```

[Argument(s)]

I/O	Argument	Description
I	ad_sel_regester_t *regester*;	Set to selected register SEL_ADANSA0 : A/D channel select register A0 (ADANSA0) SEL_ADEXICR : A/D conversion extended input control register (ADEXICR)
I	uint16_t *data*;	Set to selected register value

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_12ADC_TemperatureSensorOutput_On

Enables 12-bit A/D converter temperature sensor output circuit.

[Syntax]

```
void        R_12ADC_TemperatureSensorOutput_On ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_12ADC_TemperatureSensorOutput_Off

Disables 12-bit A/D converter temperature sensor output circuit.

[Syntax]

```
void        R_12ADC_TemperatureSensorOutput_Off ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_12ADC_InternalReferenceVoltage_On

Enables 12-bit A/D converter reference voltage circuit.

[Syntax]

```
void          R_12ADC_InternalReferenceVoltage_On ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_12ADC_InternalReferenceVoltage_Off

Disables 12-bit A/D converter reference voltage circuit.

[Syntax]

```
void        R_12ADC_InternalReferenceVoltage_Off ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_12ADC_Set_PowerOff

Halts the clock supplied to the 12-bit A/D converter.

Remark Calling this API function changes the A/D converter to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void        R_12ADC_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

Usage example

Get the A/D conversion result of 2 pins.

[GUI setting example]

12-Bit A/D convertor				
	ADC			Used
			A/D convertor operation setting	Used
			A/D conversion clock setting	PCLK
			A/D conversion mode setting	High-speed conversion
			VREF（+）setting	AVDD
			VREF（-）setting	AVSS
			Operation mode setting	Single scan mode
			Conversion start trigger setting	Software trigger
			Analog input channel setting	
			ANI00	Used
			ANI00 addition/Average function	Unused
			ANI01	Used
			ANI01 addition/Average function	Unused
			Data registers setting	
			ADconversion value addition count	1-time conversion
			Data placement	Right-alignment
			Automatic clearing	Disable automatic clearing
			ANI00 input sampling time	3.667(μs) (Actual value：3.667)
			ANI01 input sampling time	3.667(μs) (Actual value：3.667)
			A/D converted value count setting	Addition mode
			Interrupt setting	Enable AD conversion end interrupt（INTAD）
			Priority	Level 3 (Low Priority)

[API setting example]

r_cg_main.c

```
void main(void)
{
        R_MAIN_UserInit();
        /* Start user code. Do not edit comment generated here */
        /* Start AD converter */
        R_12ADC_Start();

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

r_cg_12adc_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint16_t g_12adc_ch000_value;
volatile uint16_t g_12adc_ch001_value;
/* End user code. Do not edit comment generated here */

static void __near r_12adc_interrupt(void)
{
        /* Start user code. Do not edit comment generated here */
        /* Stop AD converter */
        R_12ADC_Stop();

        /* Get AD converter result */
        R_12ADC_Get_ValueResult(ADCHANNEL0, (uint16_t *)&g_12adc_ch000_value);
        R_12ADC_Get_ValueResult(ADCHANNEL1, (uint16_t *)&g_12adc_ch001_value);
        /* End user code. Do not edit comment generated here */
}
```

3.3.48 12-bit D/A converter

Below is a list of API functions output by the Code Generator for 12-bit D/A converter use.

Table 3.50 API Functions: [12-bit D/A Converter]

API Function Name	Function
R_12DA_Create	Performs initialization necessary to control the 12-bit D/A converter.
R_12DA_Create_UserInit	Performs user-defined initialization relating to the 12-bit D/A converter.
R_12DAn_Start	Starts D/A conversion.
R_12DAn_Stop	Ends D/A conversion.
R_12DAn_Set_ConversionValue	Sets the analog voltage output to the ANO*n* pin.
R_12DA_Set_PowerOff	Halts the clock supplied to the 12-bit D/A converter.

R_12DA_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_12DA_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_12DA_Create_UserInit

Performs user-defined initialization relating to the 12-bit D/A converter.

Remark This API function is called as the R_12DA_Create callback routine.

[Syntax]

```
void        R_12DA_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_12DA*n*_Start

Starts D/A conversion.

[Syntax]

```
void        R_12DAn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_12DA*n*_Stop

Ends D/A conversion.

[Syntax]

```
void        R_12DAn_Stop ( void );
```

Remark   *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_12DA_Set_PowerOff

Halts the clock supplied to the 12-bit D/A converter.

Remark Calling this API function changes the 12-bit D/A converter to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void        R_12DA_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_12DA*n*_Set_ConversionValue

Sets the analog voltage output to the ANO*n* pin.

[Syntax]

```
#include    "r_cg_macrodriver.h"
void        R_12DAn_Set_ConversionValue ( uint16_t reg_value );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint16_t *reg_value*;	D/A conversion value.

[Return value]

None.

Usage example

Start conversion digital input to analog signal from 0x00. Add 0x10 to digital input at fixed interval. Stop conversion when digital input becomes 0xFFF.

[GUI setting example]

12-Bit D/A convertor				
	DA			Used
			D/Aconvertor operation setting	Used
			D/A reference voltage setting	AVDD/AVSS
			Data format	Right-alignment
			Use DA0	Used
			Use DA1	Unused
			D/A A/D synchronous setting	Unused

Timer array unit				
	TAU0			TAU0
		Channel 0		

[API setting example]

r_cg_main.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint16_t g_12da0_value;
/* 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 TAU0 channel 0 counter */
    R_TAU0_Channel0_Start();

    /* Set the DA0 converter value */
    g_12da0_value = 0x0000U;
    R_12DA0_Set_ConversionValue((uint8_t)g_12da0_value);

    /* Enable the DA0 converter */
    R_12DA0_Start();

    while (1U)
    {
        while (TMIF00 == 0U){
        }
        TMIF00 = 0U;

        g_12da0_value += 0x0010U;
        if (g_12da0_value <= 0x0FFFU)
        {
            /* Set the DA0 converter value */
            R_12DA0_Set_ConversionValue((uint8_t)g_12da0_value);
        }
        else
        {
            /* Stop the DA0 converter */
            R_12DA0_Stop();
        }
    }
    /* End user code. Do not edit comment generated here */
}
```

3.3.49 Operational amplifier and Analog switch

Below is a list of API functions output by the Code Generator for Operational Amplifier and Analog switch use.

Table 3.51 API Functions: [Operational amplifier and Analog switch]

API Function Name	Function
R_AMPANSW_Create	Performs initialization necessary to control the Operational amplifier and Analog switch.
R_AMPANSW_Create_UserInit	Performs user-defined initialization relating to the Operational amplifier and Analog switch.
R_OPAMPm_Set_ReferenceCircuitOn	Enables operational amplifier reference current circuit.
R_OPAMPm_Set_ReferenceCircuitOff	Disables operational amplifier reference current circuit.
R_OPAMPm_Start	Starts operational amplifier of unit *m*.
R_OPAMPm_Stop	Stops operational amplifier of unit *m*.
R_ANSW_ChargePumpm_On	Enables analog switch of unit *m*.
R_ANSW_ChargePumpm_Off	Disables analog switch of unit *m*.

R_AMPANSW_Create

Performs initialization necessary to control the Operational amplifier and Analog switch.

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

[Syntax]

```
void          R_AMPANSW_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_AMPANSW_Create_UserInit

Performs user-defined initialization relating to the Operational amplifier and Analog switch.

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

[Syntax]

```
void R_AMPANSW_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_OPAMP*m*_Set_ReferenceCircuitOn

Enables operational amplifier reference current circuit.

[Syntax]

```
void        R_OPAMPm_Set_ReferenceCircuitOn ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_OPAMP*m*_Set_ReferenceCircuitOff

Disables operational amplifier reference current circuit.

[Syntax]

```
void        R_OPAMPm_Set_ReferenceCircuitOff ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_OPAMP*m*_Start

Starts operational amplifier of unit *m*.

[Syntax]

```
void        R_OPAMPm_Start ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_OPAMP*m*_Stop

Stops operational amplifier of unit *m*.

[Syntax]

```
void        R_OPAMPm_Stop ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_ANSW_ChargePump*m*_On

Enables analog switch of unit *m*.

[Syntax]

```
void          R_ANSW_ChargePumpm_On ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_ANSW_ChargePump*m*_Off

Disables analog switch of unit *m*.

[Syntax]

```
void        R_ANSW_ChargePumpm_Off ( void );
```

Remark *m* is the unit number.

[Argument(s)]

None.

[Return value]

None.

Usage example

Perform Operational amplifer output by using analog switch.

[GUI setting example]

operational amplifier&Analog switch				
	AMPAN SW			Used
		OPAMP		Used
			Opeartional amplifer operation setting	Used
			Use operational amplifer 0	Used
			Use operational amplifer 1	Used
			Use operational amplifer 2	Used
			OPAMP0/OPAMP1 reference current circuit setting	Stop
			OPAMP0/OPAMP1 operation mode setting	Low-power mode
			OPAMP2 operation mode setting	Low-power mode
			ELC trigger setting	Operational amplifer 0: operational amplifer ELC trigger 0
				Operational amplifer 1: operational amplifer ELC trigger 1
				Operational amplifer 2: operational amplifer ELC trigger 2
			Activation/stop trigger control setting	Softwaretrigger mode
		ANSW		Used
			Analog switch operation setting	Used
			Enable analog multiplexer MUX00	Used
			Enable analog multiplexer MUX01	Used
			Enable analog multiplexer MUX02	Used
			Enable analog multiplexer MUX03	Used
			Enable analog multiplexer MUX10	Unused
			Enable analog multiplexer MUX11	Unused
			Enable analog multiplexer MUX12	Unused
			Enable analog multiplexer MUX13	Unused
			Enable low-resistance switch 0	Unused
			Enable low-resistance switch 1	Unused
			Enable low-resistance switch 2	Unused

[API setting example]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start charge pump 0 */
    R_ANSW_ChargePump0_On();

    /* Start the operational amplifier 0 */
    R_OPAMP0_Start();

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

3.3.50 Voltage reference

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

Table 3.52 API Functions: [Voltage Reference]

API Function Name	Function
R_VR_Create	Performs initialization necessary to control the Voltage reference.
R_VR_Create_UserInit	Performs user-defined initialization relating to the Voltage reference.
R_VR_Start	Enables operation of Voltage reference.
R_VR_Stop	Disables operation of Voltage reference.

R_VR_Create

Performs initialization necessary to control the Voltage reference.

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

[Syntax]

```
void        R_VR_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_VR_Create_UserInit

Performs user-defined initialization relating to the Voltage referdetector.

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

[Syntax]

```
void    R_VR_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_VR_Start

Enables operation of Voltage reference.

[Syntax]

```
void        R_VR_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_VR_Stop

Disables operation of Voltage reference.

[Syntax]

```
void        R_VR_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

Usage example

Supply the generated reference voltage to A/D convertor.

[GUI setting example]

Voltage reference				
	VR			Used
			Voltage reference operation setting	Used
			1/2 AVDD voltage output operation setting	Enable
			VREFOUT pin output level setting	1.8V

12-Bit A/Dconvertor				
	ADC			Used
			A/D convertor operation setting	Used
			A/D conversion clock setting	PCLK
			A/D conversion mode setting	High-speed conversion
			VREF（+） setting	AVREFP/VREFOUT
			VREF（-） setting	AVSS
			Operation mode setting	Single scan mode
			Conversion start trigger setting	Software trigger
			Analog input channel setting	
			ANI00	Used
			ANI00 addition/Average function	Unused
			Data registers setting	
			ADconversion value addition count	1-time conversion
			Data placement	Right-alignment
			Automatic clearing	Disable automatic clearing
			ANI00 input sampling time	3.667(μs) (Actual value：3.667)
			A/D converted value count setting	Addition mode
			Interrupt setting	Enable AD conversion end interrupt（INTAD）
			Priority	Level 3 (Low Priority)
			Data registers setting	

[API setting example]

r_cg_main.c

```
void main(void)
{
	R_MAIN_UserInit();
	/* Start user code. Do not edit comment generated here */
	/* Start VR module operation */
	R_VR_Start();

	/* Start AD converter */
	R_12ADC_Start();

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

r_cg_12adc_user.c

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

/* Start user code for global. Do not edit comment generated here */
volatile uint16_t g_12adc_ch000_value;
/* End user code. Do not edit comment generated here */

static void __near r_12adc_interrupt(void)
{
	/* Start user code. Do not edit comment generated here */
	/* Stop VR module operation */
	R_VR_Stop();

	/* Stop AD converter */
	R_12ADC_Stop();

	/* Get AD converter result */
	R_12ADC_Get_ValueResult(ADCHANNEL0, (uint16_t *)&g_12adc_ch000_value);
	/* End user code. Do not edit comment generated here */
}
```

3.3.51 Sampling output timer detector

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

Table 3.53 API Functions: [Sampling output timer detector]

API Function Name	Funciton
R_SMOTD_Create	Performinitialization necessary to control the Sampling output timer detector.
R_SMOTD_Create_UserInit	Performs user-defined initialization relating to the Sampling output timer detector.
r_smotd_counterA_interrupt	Performs processing in response to the Sampling output timer interval interrupt INTSMOTA.
r_smotd_counterB_interrupt	performs processing in response to the Sampling output timer compare match interrupt INTSMOTB.
r_smotd_smpn_interrupt	Performs processing int response to the Sampling detector detection interrupt INTSMPn.
R_SMOTD_Start	Start Starts Sampling output timer detector.
R_SMOTD_Stop	Ends Sampling output timer detector.
R_SMOTD_Set_PowerOff	Halts the clock supplied to the Sampling output timer detector.

R_SMOTD_Create

Performs initialization necessary to control the Sampling output timer detector.

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

[Syntax]

```
void        R_SMOTD_Create ( void );
```

[Argument(s)]

None.

[Return vlaue]

None.

R_SMOTD_Create_UserInit

Performs user-defined initialization relating to the Sampling output timer detector.

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

[Syntax]

```
void    R_SMOTD_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_smotd_counterA_interrupt

Performs processing in response to the Sampling output timer interval interrupt INTSMOTA.

Remark This API function is called as the interrupt process corresponding to the Sampling output timer interval interrupt INTSMOTA.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_smotd_interrupt ( void );
```

CC-RL Compiler

```
static void     __near r_smotd_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_smotd_counterB_interrupt

Performs processing in response to the Sampling output timer compare match interrupt INTSMOTB.

Remark This API function is called as the interrupt process corresponding to the Sampling output timer compare match INTSMOTB.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_smotd_counterB_interrupt ( void );
```

CC-RL Compiler

```
static void     __near r_smotd_counterB_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_smotd_smp*n*_interrupt

Performs processing in response to the Sampling detector detection interrupt INTSMP*n*.

Remark This API function is called as the interrupt process corresponding to the Sampling detector detection interrupt INTSMP*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_smotd_smpn_interrupt ( void );
```

CC-RL Compiler

```
static void    __near r_smotd_smpn_interrupt ( void );
```

Remark *n* is the sampling input number.

[Argument(s)]

None.

[Return value]

None.

R_SMOTD_Start

Starts Sampling output timer detector.

[Syntax]

```
void          R_SMOTD_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_SMOTD_Stop

Ends Sampling output timer detector.

[Syntax]

```
void        R_SMOTD_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_SMOTD_Set_PowerOff

Halts the clock supplied to the Sampling output timer detector.

[Syntax]

```
void        R_SMOTD_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.3.52 External signal sampler

Below is a list of API functions output by the Code Generator for External signal sampler use.

Table 3.54 API Functions: [External signal sampler]

API Function Name	FUnction
R_EXSD_Create	Performs initialization necessary to control the External signal sampler.
R_EXSD_Create_UserInit	Performs user-defined initialization relating to the External signal sampler.
r_exsd_interrupt	Performs processing in response to the External signal sampler edge detection interrupt INTEXSD.
R_EXSD_Start	Starts External signal sampling.
R_EXSD_Stop	Ends External signal sampling.
R_EXSD_Set_PowerOff	Halts the clock supplied to the External signal sampler.

R_EXSD_Create

Performs initialization necessary to control the External signal sampler.

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

[Syntax]

```
void        R_EXSD_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_EXSD_Create_UserInit

Performs user-defined initialization relating to the External signal sampler.

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

[Syntax]

```
void        R_EXSD_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

r_exsd_interrupt

Performs processing in response to the External signal sampler edge detection interrupt INTEXSD.

Remark This API function is called as the interrupt process corresponding to the External signal sampler edge detection interrupt INTEXSD.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_exsd_interrupt ( void );
```

CC-RL Compiler

```
static void    __near r_exsd_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_EXSD_Start

Starts A/D External signal sanpling.

[Syntax]

```
void        R_EXSD_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_EXSD_Stop

Ends External signal sampling.

[Syntax]

```
void        R_EXSD_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_EXSD_Set_PowerOff

Halts the clock supplied to the External signal sampler.

Remark Calling this API function changes the External signal sampler to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void        R_EXSD_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

3.3.53 Serial interface UARTMG

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

Figure 3.55 API Functions: [Serial interface UARTMG]

API Function Name	Function
R_UARTMGn_Create	Performs initialization necessary to control the serial interface UARTMG.
R_UARTMGn_Create_UserInit	Performs user-defined initialization related to the serial interface UARTMG.
r_uartmgn_interrupt_send	Performs processing in response to the UARTMG transmission completion interrupt INTSTMG*n*.
r_uartmgn_interrupt_receive	Performs processing in response to the UARTMG reception completion interrupt INTSRMG*n*.
r_uartmgn_interrupt_error	Perfomrs processing in response to the UARTMG reception error interrupt INTSREMG*n*.
R_UARTMGn_Start	Sets UARTMG communication to standby mode.
R_UARTMGn_Stop	Ends UARTMG communication.
R_UARTFn_Set_PowerOff	Halts the clock supplied to the serial interface UARTMG.
R_UARTMGn_Send	Starts UARTMG data transmission.
R_UARTMGn_Receive	Starts UARTMG data reception.
r_uartmgn_callback_sendend	Performs processing in response to the UARTMG transmission completion interrupt INTSTMG*n*.
r_uartmgn_callback_receiveend	Performs processing in response to the UARTMG reception completion interrupt INTSRMG*n*.
r_uartmgn_callback_error	Performs processing in response to the UARTMG reception error interrupt INTSREMG*n*.
r_uartmgn_callback_softwareoverrun	Performs processing in response to detection of overrun error.

R_UARTMG*n*_Create

Performs initialization necessary to control the serial interface UARTMG.

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

[Syntax]

```
void        R_UARTMGn_Create ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTMG*n*_Create_UserInit

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

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

[Syntax]

```
void        R_UARTMGn_Create_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartmg*n*_interrupt_send

Performs processing in response to the UARTMG transmission completion interrupt INTSTMG*n*.

Remark This API function is called as the interrupt process corresponding to the UARTMG transmission completion interrupt INTSTMG*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static void    r_uartmgn_interrupt_send ( void );
```

CC-RL Compiler

```
static void    __near r_uartmgfn_interrupt_send ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartmg*n*_interrupt_receive

Performs processing in response to the UARTMG reception completion interrupt INTSRMG*n*.

Remark This API function is called as the interrupt process corresponding to the UARMG reception completion interrupt INTSRMG*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_uartmgn_interrupt_receive ( void );
```

CC-RL Compiler

```
static void     __near r_uartmgn_interrupt_receive ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartmg*n*_interrupt_error

Performs processing in response to the UARTMG reception error interrupt INTSREMG*n*.

Remark This API function is called as the interrupt process corresponding to the UARTMG reception error interrupt INTSREMG*n*.

[Syntax]

CA78K0R Compiler

```
__interrupt static void     r_uartmgn_interrupt_error ( void );
```

CC-RL Compiler

```
static void     __near r_uartmgn_interrupt_error ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTMG*n*_Start

Sets UARTMG communication to standby mode.

[Syntax]

```
void    R_UARTMGn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTMG*n*_Stop

Ends UARTMG communication.

[Syntax]

```
void        R_UARTMGn_Stop ( void );
```

Remark   *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTMG*n*_Set_PowerOff

Halts the clock supplied to the serial interface UARTMG.

Remark Calling this API function changes the serial interface UARTMG to reset status.
For this reason, writes to the control registers after this API function is called are ignored.

[Syntax]

```
void        R_UARTMGn_Set_PowerOff ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_UARTMG*n*_Send

Starts UARTMG data transmission.

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

Remarks 2. When performing a UART transmission, R_UARTMGn_Start must be called before this API function is called.

[Syntax]

```
#include        "r_cg_macrodriver.h"
MD_STATUS       R_UARTMGn_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 specification

R_UARTMG*n*_Receive

Starts UARTMG data reception.

Remarks 1. This API function performs byte-level UART reception the number of times specified by the argument *rx_num*, and stores the data in the buffer specified by the argument *rx_buf*.

Remarks 2. Actual UART reception starts after this API function is called, and R_UARTMGn_Start is then called.

[Syntax]

```
#include        "r_cg_macrodriver.h"
MD_STATUS     R_UARTMGn_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 received data
I	uint16_t *rx_num*;	Total amount of data to receive

[Return value]

マクロ	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

r_uartmg*n*_callback_sendend

Performs processing in response to the UARTMG transmission complition interrupt INTSTMG*n*.

Remark This API function is called as the callback routine of interrupt process r_uartmgn_interrupt_send corresponding to the UARTMG transmission completion interrupt INTSTMGn (performed when number of transmission data specified by R_UARTMGn_Send argument *tx_num* has been completed).

[Syntax]

```
static void     r_uartmgfn_callback_sendend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartmgn_callback_receiveend

Performs processing in response to the UARTMG reception completion interrupt INTSRMG*n*.

Remark This API function is called as the callback routine of interrupt process r_uartmgn_interrupt_receive corresponding to the UARTMG reception completion interrupt INTSRMGn (performed when number of received data specified by R_UARTMGn_Receive argument *rx_num* has been completed).

[Syntax]

```
static void    r_uartmgn_callback_receiveend ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_uartmg*n*_callback_error

Performs processing in response to the UARTMG reception error interrupt INTSREMG*n*.

Remark This API function is called as the callback routine of interrupt process r_uartmgn_interrupt_error corresponding to the UARTMG reception error interrupt INTSREMG*n*.

[Syntax]

```
#include      "r_cg_macrodriver.h"
static void   r_uartmgn_callback_error ( uint8_t err_type );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	uint8_t *err_type*;	Trigger for UART reception error interrupt 00000xx1B: Overrn error 00000x1xB: Framing error 000001xxB: Parity error

[Return value]

None.

r_uartmg*n*_callback_softwareoverrun

Performs processing in response to detection of overrun error.

Remark This API function is called as the callback routine of interrupt process r_uartmgn_interrupt_receive corresponding to the UARTMG reception end interrupt INTSRMGn (process performed when the amount of data received is greater than the argument rx_num specified for R_UARTMGn_Receive).

[Syntax]

```
#include      "r_cg_macrodriver.h"
static void   r_uartmgn_callback_softwareoverrun ( uint16_t rx_data );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	uint16_t *rx_data*;	Receive data (greater than the argument rx_num specified for R_UARTMGn_Receive)

[Return value]

None.

3.3.54 Amplifier unit

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

Table 3.56 API Functions: [Amplifier unit]

API 関数名	機能概要
R_AMP_Create	Performs initialization necessary to control the amplifier unit.
R_AMP_Create_UserInit	Performs user-defined initialization relatede to the amplifie unit.
R_AMP_Set_PowerOn	Enables amplifier unit power supply.
R_AMP_Set_PowerOff	Disables amplifier unit power supply.
R_PGA1_Start	Starts instrumentation amplifier 1.
R_PGA1_Stop	Stops instrumentation amplifier 1.
R_AMPn_Start	Starts operational amplifier *n*.
R_AMPn_Stop	Starts operational amplifier n.Stops operational amplifier n.

R_AMP_Create

Performs initialization necessary to control the amplifier unit.

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

[Syntax]

```
void        R_AMP_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_AMP_Create_UserInit

Performs user-defined initialization relating to the amplifier unit.

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

[Syntax]

```
void        R_AMP_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_AMP_Set_PowerOn

Enables amplifier unit power supply.

[Syntax]

```
void        R_AMP_Set_PowerOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_AMP_Set_PowerOff

Disables amplifier unit power supply.

[Syntax]

```
void        R_AMP_Set_PowerOff ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA1_Start

Starts instrumentation amplifier 1.

[Syntax]

```
void        R_PGA1_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_PGA1_Stop

Stops instrumentation amplifier 1.

[Syntax]

```
void        R_PGA1_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_AMP*n*_Start

Starts operational amplifier *n*.

[Syntax]

```
void        R_AMPn_Start ( void );
```

Remark *n* is the operational amplifier unit number.

[Argument(s)]

None.

[Return value]

None.

R_AMP*n*_Stop

Stops operational amplifier *n*.

[Syntax]

```
void        R_AMPn_Stop ( void );
```

Remark *n* is the operational amplifier unit number.

[Argument(s)]

None.

[Return value]

None.

3.3.55 Data flash libraries

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

Table 3.57 API Functions: [Data Flash Libraries]

API Function Name	Function
R_FDL_Create	Performs initialization necessary to control the Data Flash Libraries.
R_FDL_Open	Starts the Data Flash Libraries.
R_FDL_Close	Stop the Data Flash Libraries.
R_FDL_Write	Writes the data to Data Flash Memories.
R_FDL_Read	Reads the data from Data Flash Memories.
R_FDL_Erase	Erases data for Data Flash Memories.

Any of the following Data Flash Libraries needs to be installed.

- **Data Flash Library Type04 for the CC-RL Compiler for RL78 Family, Japan Release**
- **Data Flash Library Type04 for the CA78K0R Compiler for RL78 Family, Japan Release**

Data Flash Library page:
https://www.renesas.com/en-us/products/software-tools/tools/self-programming-library/data-flash-libraries.html
Be sure to read the data flash library release notes before use.

Note: The code generator does not support the function that executes the following command of the data flash library. When using the following command, add a code referring to (4) and (5) of "3.3.55.1 Data flash library usage sample (CC-RL)".

- R_FDL_BLANKCHECK (Blank check command)
- R_FDL_IVERIFY (Internal verify command)

R_FDL_Create

Performs initialization necessary to control the Data Flash Libraries Type04.

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

[Syntax]

```
void        R_FDL_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_FDL_Open

Starts the Data Flash Libraries.

[Syntax]

```
void        R_FDL_Open ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_FDL_Close

Stops the Data Flash Libraries.

[Syntax]

```
void        R_FDL_Close ( void );
```

[Argument(s)]

None.

[Return value]

None.

R_FDL_Write

Writes the data to Data Flash Memories.

[Syntax]

```
pfdl_status_t    R_FDL_Write ( pfdl_u16 index, __near pfdl_u08 * buffer, pfdl_u16 bytecount );
```

[Argument(s)]

I/O	Argument	Description
I	pfdl_u16 *index*;	Writing start address of Data Flash Memories
I	pfdl_u08 * *buffer*;	Pointer to a buffer to store the write data
I	pfdl_u16 *bytecount*;	Total amount of data to wire

[Return value]

マクロ	Description
PFDL_OK	Normal completion
PFDL_BUSY	During execution of the other commands
PFDL_ERR_WRITE	Error of the writing
PFDL_ERR_PARAMETER	Error of the parameters

R_FDL_Read

Reads the data from Data Flash Memories.

[Syntax]

```
pfdl_status_t     R_FDL_Read ( pfdl_u16 index, __near pfdl_u08 * buffer, pfdl_u16 bytecount );
```

[Argument(s)]

I/O	Argument	Description
I	pfdl_u16 *index*;	Reading start address of the Data Flash Memories
I	pfdl_u08 * *buffer*;	Pointer to a buffer to store the read data
I	pfdl_u16 *bytecount*;	Total amount of data to read

[Return value]

マクロ	Description
PFDL_OK	Normal completion
PFDL_BUSY	During execution of the other commands.
PFDL_ERR_PARAMETER	Error of the parameters

R_FDL_Erase

Erases the block of Data Flash Memories.

[Syntax]

```
pfdl_status_t    R_FDL_Erase ( pfdl_u16 blockno );
```

[Argument(s)]

I/O	Argument	Description
I	pfdl_u16 *blockno*;	Erase no block for Data Flash Memories

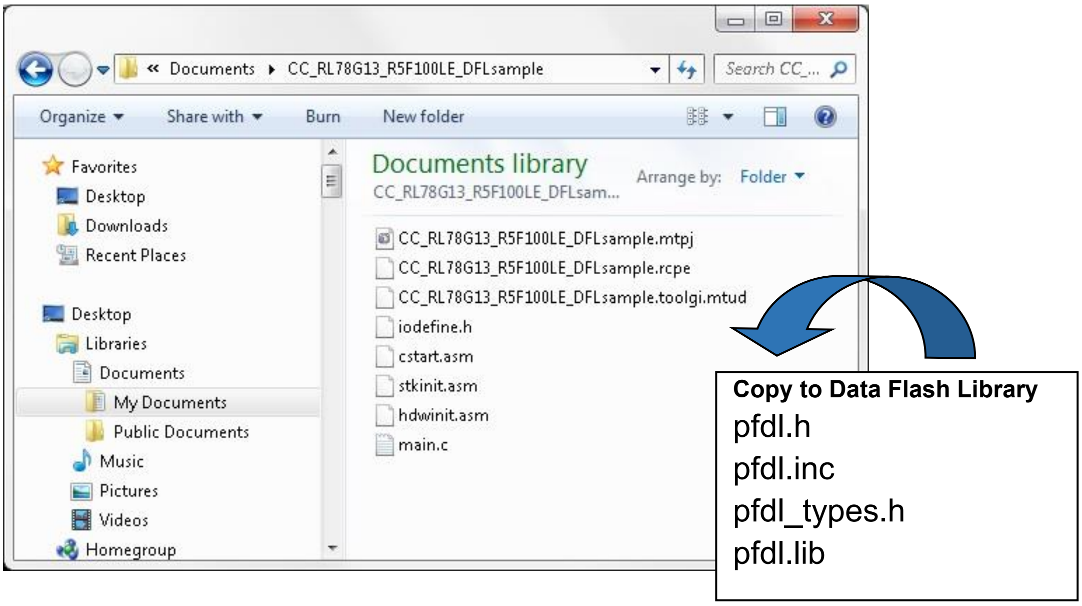
[Return value]

マクロ	Description
PFDL_OK	Normal completion
PFDL_ERR_ERASE	Error of the erasing
PFDL_ERR_PARAMETER	Error of the parameters

3.3.55.1 Data flash library usage sample (CC-RL)

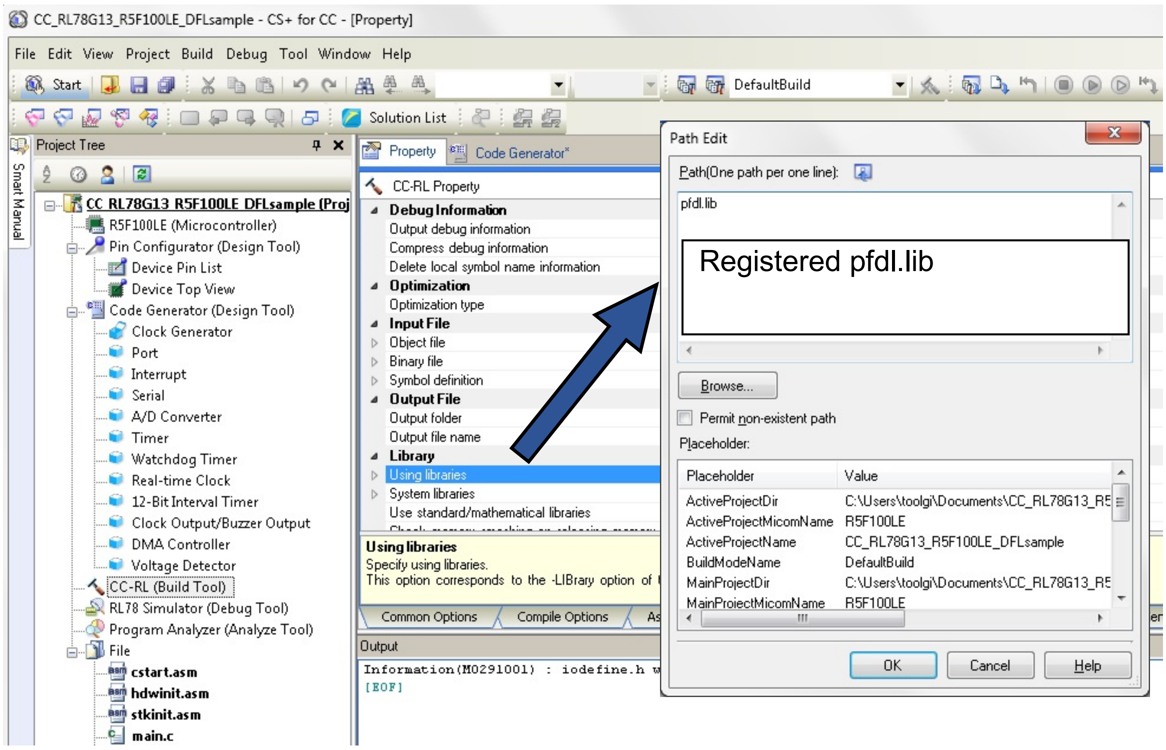
(1) Install data flash library into project

Figure 3.5 Install of flash library



(2) The Data Flash Library is registered with build tool

Figure 3.6 Register to build tool



(3) Edit r_main.c (Red is additional code)

Figure 3.7 r_main.c

```
/***********************************************************************************************************************
Pragma directive
***********************************************************************************************************************/
/* Start user code for pragma. Do not edit comment generated here */
pfdl_status_t result;
uint8_t loop;
static pfdl_u08 gtBuffer[] = { 0x11, 0x12, 0x13, 0x14, 0x55, 0xAA, 0xFF, 0x00 };
static pfdl_u08 gtReadBuffer[ 128 ];
/* End user code. Do not edit comment generated here */
    Abbreviation
void main(void)
    Abbreviation
/***********************************************************************************************************************
* Function Name: R_MAIN_UserInit
* Description   : This function adds user code before implementing main function.
* Arguments     : None
* Return Value : None
***********************************************************************************************************************/
void R_MAIN_UserInit(void)
{
    /* Start user code. Do not edit comment generated here */
    EI();
    R_FDL_Create();
    R_FDL_Open();
    /* Blank check of Data Flash */
    for ( loop=0; loop<10; loop++)
    {
       result = R_FDL_BLANKCHECK( loop * 8, gtBuffer, 8 );
       if ( result == PFDL_ERR_MARGIN )
        {
            result = R_FDL_Erase( 0 );
        }
    }
    /* Write to Data Flash */
    for ( loop=0; loop<10; loop++)
    {
        gtBuffer[ 7 ] = loop;
        result = R_FDL_Write( loop * 8, gtBuffer, 8 );
        if ( result != PFDL_OK )
        {
            break;
        }
    }
    /* Internal verification of Data Flash */
    for ( loop=0; loop<10; loop++)
    {
        result = R_FDL_IVERIFY( loop * 8, gtBuffer, 8 );
        if (result == PFDL_ERR_MARGIN )
        {
            break;
        } else {
            /* Read Data Flash */
           result = R_FDL_Read( loop * 8, &gtReadBuffer[ loop * 8], 8 );
           if ( result != PFDL_OK )
           {
               break;
           }
        }
    }
```

```
        /* Erase Data Flash */
        result = R_FDL_Erase( 0 );
        R_FDL_Close();
        /* End user code. Do not edit comment generated here */
    }
```

(4) Edit r_pfdl.h (Red is additional code)

Figure 3.8 r_cg_pfdl.h

```
/***********************************************************************************************************************
Global functions
***********************************************************************************************************************/
void R_FDL_Create(void);
pfdl_status_t R_FDL_Write(pfdl_u16 index, __near pfdl_u08* buffer, pfdl_u16 bytecount);
pfdl_status_t R_FDL_Read(pfdl_u16 index, __near pfdl_u08* buffer, pfdl_u16 bytecount);
pfdl_status_t R_FDL_Erase(pfdl_u16 blockno);
pfdl_status_t R_FDL_BLANKCHECK (pfdl_u16 index, pfdl_u16 bytecount);
pfdl_status_t R_FDL_IVERIFY (pfdl_u16 index, pfdl_u16 bytecount);
void R_FDL_Open(void);
void R_FDL_Close(void);
```

(5) Edit r_pfdl.c (Red is additional code)

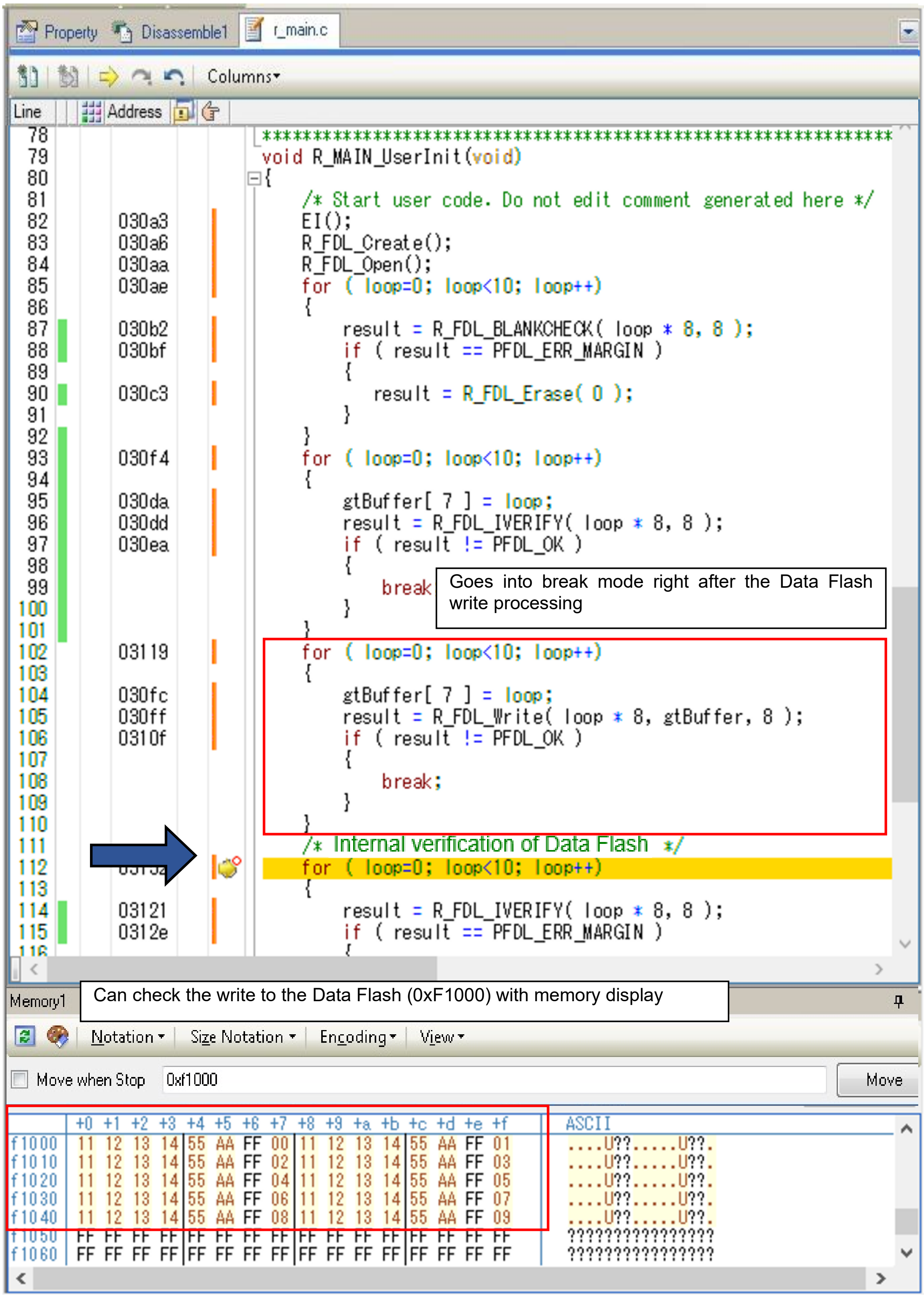
Figure 3.9 r_cg_pfdl.c

```
/***********************************************************************************************************************
* Function Name: R_FDL_Close
* Description   : This function closes the RL78 data flash library.
* Arguments     : None
* Return Value : None
***********************************************************************************************************************/
void R_FDL_Close(void)
{
    PFDL_Close();
    gFdlStatus = 0;
}

/* Start user code for adding. Do not edit comment generated here */
***********************************************************************************************************************
* Function Name: R_FDL_BLANKCHECK
* Description   : This function blank check a data to the RL78 data flash memory.
* Arguments     : index -
*                     It is destination address of Flash memory for blank check. The address range is from
*                     0x0000 to 0x0FFF
*                 buffer -
*                     The top address of data to blank check
*                 bytecount -
*                     The size of data to blank check (Unit is byte)
* Return Value : pfdl_status_t -
*                     status of blank check command
***********************************************************************************************************************/
pfdl_status_t R_FDL_BLANKCHECK(pfdl_u16 index, pfdl_u16 bytecount)
{
    if (gFdlStatus == 1)
    {
        gFdlReq.index_u16       = index;
        gFdlReq.bytecount_u16 = bytecount;
        gFdlReq.command_enu     = PFDL_CMD_BLANKCHECK_BYTES;
        gFdlResult = PFDL_Execute(&gFdlReq);
        /* Wait for completing command   */
        while(gFdlResult == PFDL_BUSY)
```

```
            {
                NOP();
                NOP();
                gFdlResult = PFDL_Handler();        /* The process for confirming end */
            }
        }
        else
        {
            gFdlResult = PFDL_ERR_PROTECTION;
        }
        return gFdlResult;
}
*******************************************************************************************************************
* Function Name: R_FDL_IVERIFY
* Description   : This function performs internal verification on the execution range area.
* Arguments      : index -
*                         It is destination address of Flash memory for iverify a data. The address range is from
*                         0x0000 to 0x0FFF
*                     buffer -
*                         The top address of data to iverify
*                     bytecount -
*                         The size of data to iverify (Unit is byte)
* Return Value : pfdl_status_t -
*                         status of iverify command
*******************************************************************************************************************/
pfdl_status_t R_FDL_IVERIFY(pfdl_u16 index, pfdl_u16 bytecount)
{
        if (gFdlStatus == 1)
        {
            gFdlReq.index_u16        = index;
            gFdlReq.bytecount_u16 = bytecount;
            gFdlReq.command_enu     = PFDL_CMD_IVERIFY_BYTES;
            gFdlResult = PFDL_Execute(&gFdlReq);
            /* Wait for completing command    */
            while(gFdlResult == PFDL_BUSY)
            {
                NOP();
                NOP();
                gFdlResult = PFDL_Handler();        /* The process for confirming end */
            }
        }
        else
        {
            gFdlResult = PFDL_ERR_PROTECTION;
        }
        return gFdlResult;
}
/* End user code. Do not edit comment generated here */
```

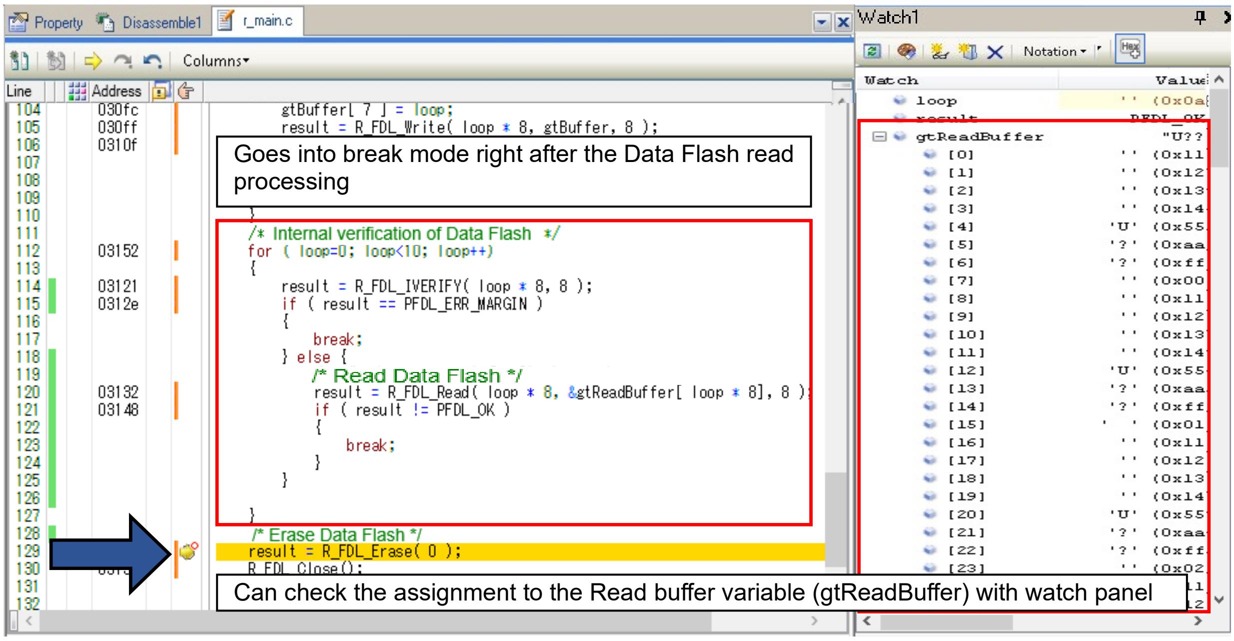
(6) Function check on the QB-R5F100LE-TB (Write)

Figure 3.10 Data flash write confirmation



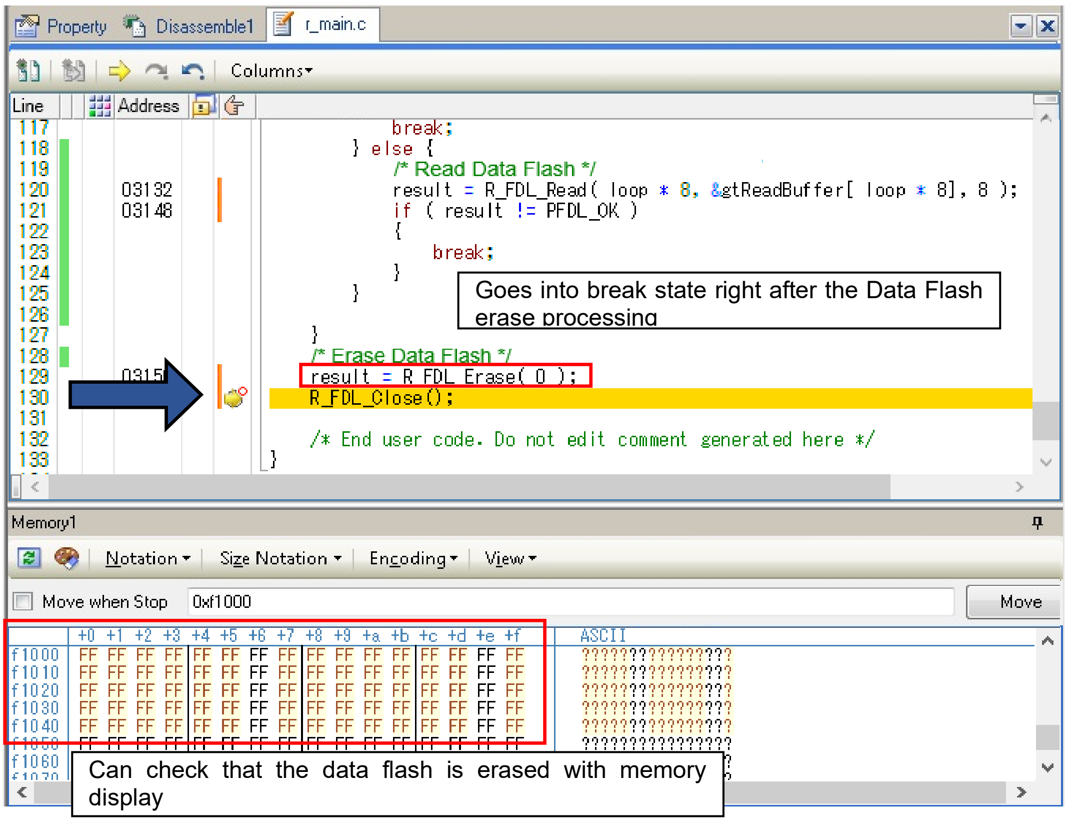
(7) Function check on the QB-R5F100LE-TB (Read)

Figure 3.11 Data flash read confirmation



(8) Function check on the QB-R5F100LE-TB (Erase)

Figure 3.12 Data flash erase confirmation



Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Jul 01, 2018		First Edition issued
1.01	Oct 01, 2019	7	Added "1.3 Complier"
		7	Added “1.4 Cautions”
		8	Added file name and API function to common of Table 2.1
		23	Added "3.2 Initialization process"
		27, 28	Added _low_level_init, HardwareSetuo
		62	Added remark to the functional outline of R_TAUm_Channeln_Start
		65	Added remark to the functional outline of R_TAUm_Channeln_Stop
		87	Added remark 3 to the functional outline of R_TMR_RJ*n*_Get_PulseWidth
		94	Added remark 3 to the functional outline of R_TMRJ*n*_Get_PulseWidth
		204	Added remark to the function outline of R_RTC_Set_CounterValue
		249	Added remark to the function outline of R_IT_Start
		250	Added remark to the function outline of R_IT_Stop
		411	Added remark 2 to the functional outline of r_csimn_callback_sendend
		412	Added remark 2 to the functional outline of r_csimn_callback_receiveend
		468	Added adr explanation (figure) to the argument remarks of R_IICAn_Master_Send
		469	Added adr explanation (figure) to the argument remarks of R_IICAn_Master_Receive
		726	Added precautions before using data flash library
		733	Added “3.2.55.1 Data flash library usage sample”
1.03	Dec 20, 2020	11, 255-262	Correction of the function names of ‘8-bit interval timer’
		8,100, 120, 121	Added following functions. R_TMRD_PWMOP_ForcedOutput_Stop R_TMRD_PWMOP_Set_PowerOff
		416	Added remark 2 to the functional outline of r_iic*mn*_interrupt
		420	Added remark 2 to the functional outline of R_IIC*mn*_Master_Send
		421	Added remark 2 to the functional outline of R_IIC*mn*_Master_Receive
		424	Added arugument description of r_iic*mn*_callback_master_error
		469	Corrected return value description of R_IICA*n*_Master_Send
		470	Corrected return value description of R_IICA*n*_Master_Receive
		471	Added remark 2 to the functional outline of r_iica*n*_callback_master_sendend
		472	Added remark 2 to the functional outline of r_iica*n*_callback_master_receiveend
1.04	Oct 15,2021	418	Added remark to the functional outline of R_IIC*mn*_StopCondition
1.05	Jul 22,2024	5-6	Updated “1.1 Overview”, “1.2 Feature” and “1.3 Compiler”
		7	Updated “Common” in Table 2.1 Output File List (1/14)
		24	Updated “3.2.2 For GNU/LLVM compiler”

Code Generator User's Manual: RL78 API Reference

Publication Date:	Rev.1.00	Jul 1, 2018
	Rev.1.01	Oct 1, 2019
	Rev.1.03	Dec 20, 2020
	Rev.1.04	Oct 15, 2021
	Rev.1.05	Jul 22, 2024

Published by: Renesas Electronics Corporation

Code Generator

Renesas Electronics Corporation