Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.

2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.

3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.

4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.

5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implants; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.

6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.

7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.

8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.

9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.

10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.

11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.

12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

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

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

(Rev.4.0-1 November 2017)
How to Use This Manual

Readers
The target readers of this manual are the application system engineers who use the Code Generator and need to understand its function.

Purpose
The purpose of this manual is to explain the user for understanding and using the Code Generator functions. We aim to help their system development including their hardware and software.

Organization
This manual can be broadly divided into the following units.
1. GENERAL
2. OUTPUT FILES
3. API FUNCTIONS

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: 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

All trademarks and registered trademarks are the property of their respective owners.
# TABLE OF CONTENTS

1. GENERAL ..................................................................................................................................................... 6
   1.1 Overview ............................................................................................................................................................ 6
   1.2 Features ............................................................................................................................................................... 6

2. OUTPUT FILES ............................................................................................................................................. 7
   2.1 Description .......................................................................................................................................................... 7

3. API FUNCTIONS ......................................................................................................................................... 17
   3.1 Overview .......................................................................................................................................................... 17
   3.2 Function Reference ........................................................................................................................................... 17
      3.2.1 Common .................................................................................................................................................. 19
      3.2.2 Clock generation circuit ........................................................................................................................ 36
      3.2.3 Voltage detection circuit (LVDA) ........................................................................................................... 44
      3.2.4 Clock frequency accuracy measurement circuit (CAC) ........................................................................... 51
      3.2.5 Low power consumption ......................................................................................................................... 61
      3.2.6 Interrupt controller (ICU) ........................................................................................................................ 72
      3.2.7 Buses ........................................................................................................................................................ 90
      3.2.8 DMA Controller (DMAC) ..................................................................................................................... 101
      3.2.9 Data transfer controller (DTC) ............................................................................................................. 113
      3.2.10 Event link controller (ELC) ................................................................................................................... 120
      3.2.11 I/O ports ................................................................................................................................................. 131
      3.2.12 Multi-function timer pulse unit 2 (MTU2) ............................................................................................ 134
      3.2.13 Multi-function timer pulse unit 3 (MTU3) ............................................................................................ 146
      3.2.14 Port output enable 2 (POE2) .................................................................................................................. 158
      3.2.15 Port output enable 3 (POE3) .................................................................................................................. 169
      3.2.16 General PWM timer (GPT) ................................................................................................................ 182
      3.2.17 16-bit timer pulse unit (TPU) ............................................................................................................... 197
      3.2.18 8-bit timer (TMR) .................................................................................................................................. 207
      3.2.19 Programmable pulse generator (PPG) ................................................................................................ 215
      3.2.20 Compare match timer (CMT) ................................................................................................................. 218
      3.2.21 Compare match timer W (CMTW) ......................................................................................................... 225
      3.2.22 Realtime clock (RTC) ......................................................................................................................... 235
      3.2.23 Watchdog timer (WDT) ....................................................................................................................... 261
      3.2.24 Independent watchdog timer (IWDT) .................................................................................................. 268
      3.2.25 Serial communications interface (SCI) ............................................................................................... 275
      3.2.26 FIFO embedded serial communications interface (SCIFA) ............................................................... 304
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.27</td>
<td>I2C bus interface (RIIC)</td>
<td>323</td>
</tr>
<tr>
<td>3.2.28</td>
<td>Serial peripheral interface (RSPI)</td>
<td>344</td>
</tr>
<tr>
<td>3.2.29</td>
<td>CRC calculator (CRC)</td>
<td>361</td>
</tr>
<tr>
<td>3.2.30</td>
<td>12-bit A/D converter (S12AD)</td>
<td>371</td>
</tr>
<tr>
<td>3.2.31</td>
<td>D/A converter (DA)</td>
<td>383</td>
</tr>
<tr>
<td>3.2.32</td>
<td>12-bit D/A converter (R12DA)</td>
<td>390</td>
</tr>
<tr>
<td>3.2.33</td>
<td>Comparator B (CMPB)</td>
<td>399</td>
</tr>
<tr>
<td>3.2.34</td>
<td>Data operation circuit (DOC)</td>
<td>407</td>
</tr>
<tr>
<td>3.2.35</td>
<td>Low power timer (LPT)</td>
<td>417</td>
</tr>
<tr>
<td>3.2.36</td>
<td>Comparator C (CMPC)</td>
<td>424</td>
</tr>
<tr>
<td>3.2.37</td>
<td>LCD controller / driver (LCD)</td>
<td>432</td>
</tr>
</tbody>
</table>

Revision Record ........................................................................................................................................ 439
1. GENERAL

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

1.1 Overview

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

1.2 Features

The Code Generator tool has the following features.

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

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

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

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

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

This appendix describes the files output by the Code Generator.

2.1 Description

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

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>File Name</th>
<th>API Function Name</th>
<th>Output (*1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common CCRX</td>
<td>r_cg_main.c</td>
<td>main</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_MAIN_UserInit</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_dbsct.c</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>r_cg_intprg.c</td>
<td>r_undefined_exception</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_privileged_exception</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_floatingpoint_exception</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_access_exception</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_nmi_exception</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_brk_exception</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_reserved_exception</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_icu_group_n_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_resetprg.c</td>
<td>PowerON_Reset</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PowerON_Reset_PC</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_sbrk.c</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>r_cg_vecttbl.c</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>r_cg_sbrk.h</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>r_cg_stackact.h</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>r_cg_vect.h</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>r_cg_hardware_setup.c</td>
<td>HardwareSetup</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Systeminit</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_macrodriver.h</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>r_cg_userdefine.h</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Common IAR(ICCRX)</td>
<td>r_cg_main.c</td>
<td>main</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_MAIN_UserInit</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_intprg.c</td>
<td>r_brk_exception</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_icu_group_n_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_NMI_handler</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_systeminit.c</td>
<td>R_Systeminit</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_low_level_init</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_macrodriver.h</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>r_cg_userdefine.h</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>
### Table 2.2 Output File List (2/10)

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>File Name</th>
<th>API Function Name</th>
<th>output (*1)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Common GNU(GCCRX)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_main.c</td>
<td>main</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_MAIN_UserInit</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_vector_table.c</td>
<td>r_undefined_exception</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_reserved_exception</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_nmi_exception</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_brk_exception</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_icu_group_n_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_reset_program.asm</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>r_cg_interrupt_handlers.h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>r_cg_hardware_setup.c</td>
<td>HardwareSetup</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Systeminit</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_macrodriver.h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>r_cg_userdefine.h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Clock generation circuit</strong></td>
<td>r_cg_cgc.c</td>
<td>R_CGC_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_CGC_Set_ClockMode</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>r_cg_cgc_user.c</td>
<td>R_CGC_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_cgc_oscillation_stop_nmi_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_cgc_oscillation_stop_interupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_cgc.h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Voltage detection circuit (LVDA)</strong></td>
<td>r_cg_lvd.c</td>
<td>R_LVDn_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_LVDn_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_LVDn_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_lvd_user.c</td>
<td>R_LVDn_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_lvd_lvdn_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_lvd.h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Clock frequency accuracy measure- ment circuit (CAC)</strong></td>
<td>r_cg_cac.c</td>
<td>R_CAC_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_CAC_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_CAC_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_cac_user.c</td>
<td>R_CAC_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_cac_mendf_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_cac_ferrf_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_cac_ovff_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_cac.h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Low power consumption</strong></td>
<td>r_cg_lpc.c</td>
<td>R_LPC_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_LPC_AllModuleClockStop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_LPC_ChangeSleepModeReturnClock</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_LPC_Sleep</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_LPC_DeepSleep</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_LPC_DeepSoftWareStandby</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_LPC_SoftwareStandby</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_LPC_ChangeOperatingPowerControl</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_lpc_user.c</td>
<td>R_LPC_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>r_cg_lpc.h</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Table 2.3  Output File List (3/10)

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>File Name</th>
<th>API Function Name</th>
<th>Output (*1)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Interrupt controller (ICU)</strong></td>
<td>r_cg_icu.c</td>
<td>R_ICU_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ICU_IRQn_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ICU_IRQn_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ICU_Software_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ICU_Software2_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ICU_Software_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ICU_Software2_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ICU_SoftwareInterrupt_Generate</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ICU_SoftwareInterrupt2_Generate</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_icu_user.c</td>
<td>R_ICU_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_icu_irqn_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_icu_software_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_icu_software2_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_icu_nmi_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_icu.h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Buses</strong></td>
<td>r_cg_bsc.c</td>
<td>R_BSC_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_BSC_Error_Monitoring_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_BSC_Error_Monitoring_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_BSC_InitializeSDRAM</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_bsc_user.c</td>
<td>R_BSC_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_bsc_buserr_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_bsc.h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>DMA Controller(DMAC)</strong></td>
<td>r_cg_dmac.c</td>
<td>R_DMA_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DMAcN_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DMAcN_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DMAcN_Set_SoftwareTrigger</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DMAcN_Clear_SoftwareTrigger</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_dmac_user.c</td>
<td>r_dmac_dmaclinterrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_dmacl_callback_transfer_end</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_dmacl_callback_transfer_escape_end</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_DMA_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>r_cg_dmac.h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Data transfer controller (DTC)</strong></td>
<td>r_cg_dtc.c</td>
<td>R_DTC_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DTCm_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DTCm_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_dtc_user.c</td>
<td>R_DTC_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>r_cg_dtc.h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>File Name</td>
<td>API Function Name</td>
<td>output (*1)</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>--------------------</td>
<td>----------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Event link controller (ELC)</td>
<td>r_cg_elc.c</td>
<td>R_ELC_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ELC_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ELC_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ELC_GenerateSoftwareEvent</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ELC_Set_PortBuffer</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ELC_Get_PortBuffer</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_elc_user.c</td>
<td>R_ELC_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_elc_elsri_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_elc.h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>I/O ports</td>
<td>r_cg_port.c</td>
<td>R_PORT_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_port_user.c</td>
<td>R_PORT_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>r_cg_port.h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Multi-function timer pulse unit 2 (MTU2)</td>
<td>r_cg_mtu2.c</td>
<td>R_MTU2_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_MTU2_Cn_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_MTU2_Cn_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_mtu2_user.c</td>
<td>R_MTU2_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_mtu2_tgimn_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_mtu2_cj_tgimn_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_mtu2_tcivn_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_mtu2_cj_tcivn_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_mtu2.h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Multi-function timer pulse unit 3 (MTU3)</td>
<td>r_cg_mtu3.c</td>
<td>R_MTU3_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_MTU3_Cn_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_MTU3_Cn_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_mtu3_user.c</td>
<td>R_MTU3_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_mtu3_tgimn_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_mtu3_cj_tgimn_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_mtu3_tcivn_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_mtu3_cj_tcivn_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_mtu3.h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Port output enable 2 (POE2)</td>
<td>r_cg_poe2.c</td>
<td>R_POE2_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_POE2_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_POE2_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_POE2_Set_HiZ_MTUn</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_POE2_Clear_HiZ_MTUn</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_poe2_user.c</td>
<td>R_POE2_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_poe2_oein_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_poe2.h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>File Name</td>
<td>API Function Name</td>
<td>output (*1)</td>
</tr>
<tr>
<td>----------------------------------------------------------</td>
<td>------------------------------------</td>
<td>------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Port output enable 3 (POE3)</td>
<td>r_cg_poe3.c</td>
<td>R_POE3_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_POE3_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_POE3_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_POE3_Set_HiZ_MTUn</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_POE3_Clear_HiZ_MTUn</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_POE3_Set_HiZ_GPTn</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_POE3_Clear_HiZ_GPTn</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_poe3_user.c</td>
<td>R_POE3_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_poe3_oein_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_poe3.h</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>General PWM timer (GPT)</td>
<td>r_cg_gpt.c</td>
<td>R_GPT_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_GPTn_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_GPTn_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_GPTn_HardwareStart</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_GPTn_HardwareStop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_gpt_user.c</td>
<td>R_GPT_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_gpt_gtcimn_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_gpt_gtcivn_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_gpt_gtcuin_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_gpt_gdten_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_gpt_etgip_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_gpt_etgin_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_gpt.h</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>16-bit timer pulse unit (TPU)</td>
<td>r_cg_tpu.c</td>
<td>R_TPU_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_TPUn_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_TPUn_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_tpu_user.c</td>
<td>R_TPU_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_tpu_tgimm_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_tpu_tcbn_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_tpu_tcbun_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_tpu.h</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>8-bit timer (TMR)</td>
<td>r_cg_tmr.c</td>
<td>R_TMR_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_TMRn_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_TMRn_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_tmr_user.c</td>
<td>R_TMR_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_tmr_cmimm_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_tmr_cmimmn_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_tmr.h</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Programmable pulse generator (PPG)</td>
<td>r_cg_ppg.c</td>
<td>R_PPG_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_ppg_user.c</td>
<td>R_PPG_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>r_cg_ppg.h</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>
### Table 2.6 Output File List (6/10)

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>File Name</th>
<th>API Function Name</th>
<th>Output (*1)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compare match timer (CMT)</strong></td>
<td>r_cg_cmt.c</td>
<td>R_CMTn_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_CMTn_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_CMTn_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_cmt_user.c</td>
<td>R_CMTn_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_cmt_cmin_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_cmt.h</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td><strong>Compare match timer W (CMTW)</strong></td>
<td>r_cg_cmtw.c</td>
<td>R_CMTWn_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_CMTWn_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_CMTWn_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_cmtw_user.c</td>
<td>R_CMTWn_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_cmtw_cmwin_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_cmtw_icmin_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_cmtw_ocmin_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_cmtw.h</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td><strong>Realtime clock (RTC)</strong></td>
<td>r_cg_RTC.c</td>
<td>R_RTC_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RTC_Set_CalendarAlarm</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RTC_Set_BinaryAlarm</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RTC_Set_ConstPeriodInterruptOn</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RTC_Set_ConstPeriodInterruptOff</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RTC_Set_CarryInterruptOn</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RTC_Set_CarryInterruptOff</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RTC_Set_RTCOUTOn</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RTC_Set_RTCOUTOff</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RTC_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RTC_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RTC_Restart</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RTC_Set_CalendarCounterValue</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RTC_Get_CalendarCounterValue</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RTC_Set_BinaryCounterValue</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RTC_Get_BinaryCounterValue</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RTC_Get_CalendarTimeCaptureValue</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RTC_Get_BinaryTimeCaptureValue</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_RTC_user.c</td>
<td>R_RTC_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rtc alm_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rtc prd interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>rtc cup interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_RTC.h</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>
## Table 2.7 Output File List (7/10)

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>File Name</th>
<th>API Function Name</th>
<th>output (*1)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Watchdog timer (WDT)</strong></td>
<td>r_cg_wdt.c</td>
<td>R_WDT_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_WDT_Restart</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_wdt_user.c</td>
<td>R_WDT_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_wdt_nmi_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_wdt_wuni_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_wdt.h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Independent watchdog timer (IWDT)</strong></td>
<td>r_cg_iwdt.c</td>
<td>R_IWDT_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_IWDT_Restart</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_iwdt_user.c</td>
<td>R_IWDT_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_iwdt_nmi_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_iwdt_iwuni_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_iwdt.h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Serial communications interface (SCI)</strong></td>
<td>r_cg_sci.c</td>
<td>R_SCIn_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_SCIn_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_SCIn_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_SCIn_Serial_Send</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_SCIn_Serial_Receive</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_SCIn_Serial_Multiprocessor_Send</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_SCIn_Serial_Multiprocessor_Receive</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_SCIn_Serial_Send_Receive</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_SCIn_SmartCard_Send</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_SCIn_SmartCard_Receive</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_SCIn_IIC_Master_Send</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_SCIn_IIC_Master_Receive</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_SCIn_SPI_Master_Send</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_SCIn_SPI_Master_Send_Receive</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_sci_user.c</td>
<td>R_SCIn_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>r_scin_transmitend_interrupt</td>
<td></td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_scin_transmit_interrupt</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>r_scin_receive_interrupt</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>r_scin_receiveerror_interrupt</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>r_scin_callback_transmitend</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>r_scin_callback_receiveend</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>r_scin_callback_receiveerror</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>r_cg_sci.h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>File Name</td>
<td>API Function Name</td>
<td>output (*1)</td>
</tr>
<tr>
<td>----------------------------------------------------------</td>
<td>------------------------</td>
<td>------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>FIFO embedded serial communications interface (SCIFA)</td>
<td>r_cg_scifa.c</td>
<td>R_SCIFAn_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_SCIFAn_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_SCIFAn_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_SCIFAn_Serial_Send</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_SCIFAn_Serial_Receive</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_SCIFAn_Serial_Send_Receive</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_scifa_user.c</td>
<td>R_SCIFAn_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_scifan_tEIF_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_scifan_xEIF_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_scifan_erIF_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_scifan_bRF_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_scifan_dRF_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_scifan_callback_transmit</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_scifan_callback_receive</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_scifan_callback_error</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_scifa.h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>I2C bus interface (RIIC)</td>
<td>r_cg_rlic.c</td>
<td>R_RIIcn_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RIIcn_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RIIcn_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RIIcn_Master_Send</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RIIcn_Master_Receive</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RIIcn_Slave_Send</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RIIcn_Slave_Receive</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RIIcn_StartCondition</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RIIcn_StopCondition</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_rlic_user.c</td>
<td>R_RIIcn_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_rlicn_error_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_rlicn_receive_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_rlicn_transmit_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_rlicn_transmitend_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_rlicn_callback_receiveerror</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_rlicn_callback_transmit</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_rlicn_callback_receiveerror</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_rlic.h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>File Name</td>
<td>API Function Name</td>
<td>Output (*1)</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>----------------------------</td>
<td>-------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Serial peripheral interface (RSPI)</td>
<td>r_cg_rspi.c</td>
<td>R_RSPIn_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RSPIn_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RSPIn_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RSPIn_Send</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_RSPIn_Send_Receive</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_rspi_user.c</td>
<td>R_RSPIn_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_rspin_receive_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_rspin_transmit_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_rspin_error_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_rspin_idle_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_rspin_callback_receiveend</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_rspin_callback_error</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_rspin_callback_transmitend</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_rspi.h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CRC calculator (CRC)</td>
<td>r_cg_crc.c</td>
<td>R_CRC_SetCRC8</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_CRC_SetCRC16</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_CRC_SetCCITT</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_CRC_SetCRC32</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_CRC_SetCRC32C</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_CRC_Input_Data</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_CRC_Get_Result</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_crc.h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12-bit A/D converter (S12AD)</td>
<td>r_cg_s12ad.c</td>
<td>R_S12ADn_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_S12ADn_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_S12ADn_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_S12ADn_Get_ValueResult</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_S12ADn_Set_CompareValue</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_s12ad_user.c</td>
<td>R_S12ADn_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>s12adn_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>s12adn_groupb_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>s12adn_compare_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_s12ad.h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D/A converter (DA)</td>
<td>r_cg_da.c</td>
<td>R_DA_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DAm_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DAm_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DAm_Set_ConversionValue</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_da_user.c</td>
<td>R_DA_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>r_cg_da.h</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 2.10 Output File List (10/10)

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>File Name</th>
<th>API Function Name</th>
<th>output (*1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12-bit converter (R12DA)</td>
<td>r_cg_r12da.c</td>
<td>R_R12DA_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_R12DAn_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_R12DAn_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_R12DAn_Set_ConversionValue</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_R12DA_Sync_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_R12DA_Sync_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_r12da_user.c</td>
<td>R_R12DA_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>r_cg_r12da.h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Comparator B (CMPB)</td>
<td>r_cg_cmpb.c</td>
<td>R_CMPB_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_CMPBn_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_CMPBn_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_cmpb_user.c</td>
<td>R_CMPB_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_cmpb_cmpbn_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_cmpb.h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Data operation circuit (DOC)</td>
<td>r_cg_doc.c</td>
<td>R_DOC_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DOC_SetMode</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DOC_WriteData</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DOC_GetResult</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DOC_ClearFlag</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_doc_user.c</td>
<td>R_DOC_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_doc_dopcf_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_doc.h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Low power timer (LPT)</td>
<td>r_cg_lpt.c</td>
<td>R_LPT_Create</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_LPT_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_LPT_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_lpt_user.c</td>
<td>R_LPT_Create_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>r_cg_lpt.h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Comparator C (CMPC)</td>
<td>r_cg_cmpc.c</td>
<td>R_CMPC_CREATE</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_CMPCn_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_CMPCn_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_cmpc_user.c</td>
<td>R_CMPC_CREATE_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_cmpc_cmpcn_interrupt</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_cmpc.h</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>LCD controller / driver (LCD)</td>
<td>r_cg_lcd.c</td>
<td>R_LCD_CREATE</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_LCD_Start</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_LCD_Stop</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_LCDVoltage_On</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_LCDVoltage_Off</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>r_cg_lcd_user.c</td>
<td>R_LCD_CREATE_UserInit</td>
<td>M</td>
</tr>
<tr>
<td></td>
<td>r_cg_lcd.h</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

*1 In case of [API output control] setting are default ([Output all API functions according to the setting]).

A : Output by settings on each peripheral functions panel automatically.

M : Output by the file used setting in API property.
3. API FUNCTIONS
This appendix describes the API functions output by the Code Generator.

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

- Macro names are in ALL CAPS. The number in front of the macro name is a hexadecimal value; this is the same value as the macro value.
- Local variable names are in all lower case.
- Global variable names start with a "g" and use Camel Case.
- Names of pointers to global variables start with a "gp" and use Camel Case.
- Names of elements in enum statements are in ALL CAPS.

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

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

Figure 3.1 Notation Format of API Functions

(1)       (2)       (3)       (4)       (5)
(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.

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>(b)</td>
<td>(c)</td>
</tr>
</tbody>
</table>

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

(b) **Argument**
Argument data type

(c) **Description**
Description of argument

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

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>(b)</td>
</tr>
</tbody>
</table>

(a) **Macro**
Macro of return value

(b) **Description**
Description of return value
3.2.1 Common

Below is a list of API functions output by the Code Generator for common use.
Performs processing in response to the exception (other than undefined instruction exception, reset, non-maskable interrupt and unconditional trap).

Table 3.1 API Functions: [Common]

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

Performs processing in response to the undefined instruction exception.

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
PowerON_Reset

Performs processing in response to the reset.

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
PowerON_Reset_PC

Performs processing in response to the reset.

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

[Syntax]

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

[Argument(s)]
None.

[Return value]
None.
**r_privileged_exception**

Performs processing in response to the privileged instruction exception.

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_floatingpoint_exception**

Performs processing in response to the floating-point exception.

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_access_exception

Performs processing in response to the access exception.

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

[Syntax]

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

[Argument(s)]
None.

[Return value]
None.
r_nmi_exception

Performs processing in response to the non-maskable interrupt.

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

[Syntax]
void r_nmi_exception ( void );

[Argument(s)]
None.

[Return value]
None.
r_brk_exception

Performs processing in response to the unconditional trap.

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

[Syntax]

void    r_brk_exception ( void );

[Argument(s)]

None.

[Return value]

None.
**r_reserved_exception**

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

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
HardwareSetup

Performs initialization necessary to control the various hardwares.

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

[Syntax]

    void HardwareSetup ( void );

[Argument(s)]

    None.

[Return value]

    None.
**R_Systeminit**

Performs initialization necessary to control the various peripheral functions.

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
This is a main function.
Remark This API function is called as the PowerON_Reset callback routine.

[Syntax]
void main ( void );

[Argument(s)]
None.

[Return value]
None.
**R_MAIN_UserInit**

Performs user-defined initialization.

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_icu_group_n_interrupt

Performs processing in response to the group interrupts.

[Syntax]

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

Remark  
$n$ is the group interrupt number.

[Argument(s)]

None.

[Return value]

None.
**_NMI_handler**

Performs processing in response to the non-maskable interrupt.

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
_low_level_init

Performs initialization necessary to control the various hardwares.

[Syntax]

void _low_level_init ( void );

[Argument(s)]
None.

[Return value]
None.
3.2.2 Clock generation circuit

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

Table 3.2 API Functions: [Clock Generation Circuit]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_CGC_Create</td>
<td>Performs initialization required to control the clock generation circuit.</td>
</tr>
<tr>
<td>R_CGC_Create_UserInit</td>
<td>Performs user-defined initialization relating to the clock generation circuit.</td>
</tr>
<tr>
<td>r_cgc_oscillation_stop_interrupt</td>
<td>Performs processing in response to the oscillation stop detection interrupt.</td>
</tr>
<tr>
<td>r_cgc_oscillation_stop_nmi_interrupt</td>
<td>Performs processing in response to the oscillation stop detection NMI.</td>
</tr>
<tr>
<td>R_CGC_Set_ClockMode</td>
<td>Sets the clock source.</td>
</tr>
</tbody>
</table>
**R_CGC_Create**

Performs initialization required to control the clock generation circuit.

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_CGC_Create_UserInit

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

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

[Syntax]

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

[Argument(s)]
None.

[Return value]
None.
r_cgc_oscillation_stop_interrupt
Performs processing in response to the oscillation stop detection interrupt.

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

[Syntax]
```c
static void r_cgc_oscillation_stop_interrupt ( void );
```

[Argument(s)]
None.

[Return value]
None.
**r_cgc_oscillation_stop_nmi_interrupt**

Performs processing in response to the oscillation stop detection NMI.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
R_CGC_Set_ClockMode

Sets the clock source.

[Syntax]

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

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>clock_mode_t mode;</td>
<td>Clock source type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAINCLK : Main clock oscillator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBCLK : Sub-clock oscillator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PLLCLK : PLL circuit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HOCO : High-speed on-chip oscillator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LOCO : Low-speed on-chip oscillator</td>
</tr>
</tbody>
</table>

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ERROR1</td>
<td>Exit with error (abend)</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument mode specification</td>
</tr>
</tbody>
</table>
Usage example

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

[GUI setting example]

<table>
<thead>
<tr>
<th>Clock Generator</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>CGC</td>
<td>Used</td>
</tr>
<tr>
<td>Main clock oscillator</td>
<td>Forced oscillation</td>
</tr>
<tr>
<td>Main clock oscillation source</td>
<td>Resonator</td>
</tr>
<tr>
<td>Main clock oscillation source Frequency</td>
<td>24(MHz)</td>
</tr>
<tr>
<td>Oscillator wait time</td>
<td>11000(μs)</td>
</tr>
<tr>
<td>Oscillation stop detection function</td>
<td>Enabled (with interrupt request output)</td>
</tr>
<tr>
<td>OSTD1 Priority</td>
<td>Level 15 (highest)</td>
</tr>
<tr>
<td>PLL Operation</td>
<td>Unused</td>
</tr>
<tr>
<td>SubCLK Operation</td>
<td>Unused</td>
</tr>
<tr>
<td>HOCO Operation</td>
<td>Used</td>
</tr>
<tr>
<td>HOCO Frequency</td>
<td>16 (MHz)</td>
</tr>
<tr>
<td>LOCO Operation</td>
<td>Used</td>
</tr>
<tr>
<td>LOCO Frequency</td>
<td>240 (kHz)</td>
</tr>
<tr>
<td>IWDT Operation</td>
<td>Unused</td>
</tr>
<tr>
<td>Clock source</td>
<td>Main clock oscillator</td>
</tr>
<tr>
<td>System clock (ICLK)</td>
<td>x 1/4 6 (MHz)</td>
</tr>
<tr>
<td>Peripheral module clock (PCLKA)</td>
<td>x 1/4 6 (MHz)</td>
</tr>
<tr>
<td>Peripheral module clock (PCLKB)</td>
<td>x 1/4 6 (MHz)</td>
</tr>
<tr>
<td>Peripheral module clock (PCLKC)</td>
<td>x 1/4 6 (MHz)</td>
</tr>
<tr>
<td>Peripheral module clock (PCLKD)</td>
<td>x 1/4 6 (MHz)</td>
</tr>
<tr>
<td>External bus clock (BCLK)</td>
<td>x 1/4 6 (MHz)</td>
</tr>
<tr>
<td>Flash IF clock (FCLK)</td>
<td>x 1/4 6 (MHz)</td>
</tr>
<tr>
<td>USB clock (UCLK)</td>
<td>x 1/3 8 (MHz)</td>
</tr>
<tr>
<td>RTC clock setting</td>
<td>Unused</td>
</tr>
<tr>
<td>BCLK Operation</td>
<td>Unused</td>
</tr>
<tr>
<td>SDCLK Operation</td>
<td>Unused</td>
</tr>
<tr>
<td>Debug interface setting</td>
<td>Unused</td>
</tr>
</tbody>
</table>
[API setting example]

r_cg_main.c

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

r_cg_cgc_user.c

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

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

Table 3.3 API Functions: [Voltage Detection Circuit]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_LVDn_Create</td>
<td>Performs initialization necessary to control the voltage detection circuit.</td>
</tr>
<tr>
<td>R_LVDn_Create_UserInit</td>
<td>Performs user-defined initialization relating to the voltage detection circuit.</td>
</tr>
<tr>
<td>r_lvd_lvdn_interrupt</td>
<td>Performs processing in response to the voltage monitoring interrupt.</td>
</tr>
<tr>
<td>R_LVDn_Start</td>
<td>Starts voltage monitoring (when in interrupt mode, and interrupt &amp; reset mode).</td>
</tr>
<tr>
<td>R_LVDn_Stop</td>
<td>Ends voltage monitoring (when in interrupt mode, and interrupt &amp; reset mode).</td>
</tr>
</tbody>
</table>
**R_LVDn_Create**

Performs initialization necessary to control the voltage detection circuit.

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

**[Syntax]**

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

**Remark** $n$ is the circuit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_LVDn_Create_UserInit**

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

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

**[Syntax]**

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

Remark  \( n \) is the circuit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_lvd_lvdn_interrupt**

Performs processing in response to the voltage monitoring \( n \) interrupt.

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

**Syntax**

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

**Remark**  \( n \) is the circuit number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_LVDn_Start

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

[Syntax]

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

Remark  \( n \) is the circuit number.

[Argument(s)]

None.

[Return value]

None.
R_LVDn_Stop

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

[Syntax]

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

Remark $n$ is the circuit number.

[Argument(s)]

None.

[Return value]

None.
Usage example

Perform software reset at voltage monitoring interrupt.

[GUI setting example]

<table>
<thead>
<tr>
<th>Voltage Detection Circuit</th>
<th>LVD1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voltage detection 1 circuit operation setting</td>
<td>Used</td>
</tr>
<tr>
<td>Enable voltage monitoring digital filter</td>
<td>Unused</td>
</tr>
<tr>
<td>Voltage detection level</td>
<td>2.85(V)</td>
</tr>
<tr>
<td>Voltage monitoring circuit mode</td>
<td>Voltage monitoring 1 interrupt enabled when Vdet1 is crossed</td>
</tr>
<tr>
<td>Interrupt setting</td>
<td>Maskable interrupt</td>
</tr>
<tr>
<td>Interrupt ELC event generation condition</td>
<td>When VCC &lt; Vdet1 (drop) is detected</td>
</tr>
<tr>
<td>Priority</td>
<td>Level 15 (highest)</td>
</tr>
</tbody>
</table>

[API setting example]

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

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

```c
r_cg_lvd_user.c
static void r_lvd_lvd1_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Software reset */
    SYSTEM.PRCR.WORD = 0xA502;
    SYSTEM.SWRR = 0xA501;
    /* End user code. Do not edit comment generated here */
}
```
3.2.4 Clock frequency accuracy measurement circuit (CAC)

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

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_CAC_Create</td>
<td>Performs initialization necessary to control the clock frequency accuracy measurement circuit.</td>
</tr>
<tr>
<td>R_CAC_Create_UserInit</td>
<td>Performs user-defined initialization relating to the clock frequency accuracy measurement circuit.</td>
</tr>
<tr>
<td>r_cac_mendf_interrupt</td>
<td>Performs processing in response to the measurement end interrupt.</td>
</tr>
<tr>
<td>r_cac_ferrf_interrupt</td>
<td>Performs processing in response to the frequency error interrupt.</td>
</tr>
<tr>
<td>r_cac_ovff interrupt</td>
<td>Performs processing in response to the overflow interrupt.</td>
</tr>
<tr>
<td>R_CAC_Start</td>
<td>Starts measurement of the accuracy of the clock frequency.</td>
</tr>
<tr>
<td>R_CAC_Stop</td>
<td>Ends measurement of the accuracy of the clock frequency.</td>
</tr>
</tbody>
</table>
R_CAC_Create

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

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

[Syntax]
void R_CAC_Create ( void );

[Argument(s)]
None.

[Return value]
None.
Perform user-defined initialization relating to the clock frequency accuracy measurement circuit.

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

**Syntax**

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

**Argument(s)**

None.

**Return value**

None.
r_cac_mendf_interrupt

Performs processing in response to the measurement end interrupt.

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

[Syntax]

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

[Argument(s)]
None.

[Return value]
None.
**r_cac_ferrf_interrupt**

Performs processing in response to the frequency error interrupt.

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

**[Syntax]**

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

**[Argument(s)]**
None.

**[Return value]**
None.
**r_cac_ovff_interrupt**

Performs processing in response to the overflow interrupt.

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

**[Syntax]**

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

**[Argument(s)]**
None.

**[Return value]**
None.
R_CAC_Start

Starts measurement of the accuracy of the clock frequency.

[Syntax]

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

[Argument(s)]
None.

[Return value]
None.
R_CAC_Stop

Ends measurement of the accuracy of the clock frequency.

[Syntax]
void    R_CAC_Stop ( void );

[Argument(s)]
None.

[Return value]
None.
Usage example

Count frequency error.

[GUI setting example]

<table>
<thead>
<tr>
<th>Clock</th>
<th>Frequency Accuracy</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measurement Circuit</td>
<td>CAC</td>
<td>Used</td>
</tr>
</tbody>
</table>

| CAC operation setting | Measured reference setting Clock select | Main clock |
| Measurement reference setting Frequency | x 1/32 (0.75)(MHz) |

| Valid edge | Rising |
| Measurement target setting Clock select | Main clock |
| Measurement target setting Frequency | x 1 (24)(MHz) |

| Maximum positive deviation | 5%(Actual value : 3.125) |
| Maximum negative deviation | 5%(Actual value : 6.25) |

| Enable frequency error interrupt (FERRF) | Used |
| Priority | (Group BL0)(FERRF) | Level 15 (highest) |

| Enable measurement end interrupt (MENDF) | Used |
| Priority | (Group BL0)(MENDF) | Level 15 (highest) |

| Enable overflow interrupt (OVFF) | Used |
| Priority | (Group BL0)(OVFF) | Level 15 (highest) |

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

| Group | Used |
[API setting example]

r_cg_main.c

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

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

r_cg_cac_user.c

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

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

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

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

Table 3.5 API Functions: [Low Power Consumption]

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

Performs initialization required to control the low power consumption.

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

[Syntax]

    void R_LPC_Create ( void );

[Argument(s)]

    None.

[Return value]

    None.
**R_LPC_Create_UserInit**

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

**Remark**  This API function is called as the _R_LPC_Create_ callback routine.

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_LPC_AllModuleClockStop**

Stops the clock for all modules.

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ERROR1</td>
<td>Exit with error (abend)</td>
</tr>
</tbody>
</table>
R_LPC_ChangeSleepModeReturnClock

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

[Syntax]

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

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>return_clock_t</td>
<td>Clock source type</td>
</tr>
<tr>
<td></td>
<td>clock;</td>
<td>RETURN_LOCO : Low-speed on-chip oscillator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RETURN_HOCO : High-speed on-chip oscillator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RETURN_MAIN_CLOCK : Main clock oscillator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RETURN_DISABLE : Switching of the clock source</td>
</tr>
<tr>
<td></td>
<td></td>
<td>does not proceed.</td>
</tr>
</tbody>
</table>

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ERROR1</td>
<td>Change to the low-speed operating mode ended abnormally.</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument clock specification</td>
</tr>
</tbody>
</table>
R_LPC_Sleep

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ERROR1</td>
<td>Exit with error (abend)</td>
</tr>
</tbody>
</table>
R_LPC_DeepSleep

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

[Syntax]
```c
#include "r_cg_macrodriver.h"
MD_STATUS R_LPC_DeepSleep ( void );
```

[Argument(s)]
None.

[Return value]
<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ERROR1</td>
<td>Exit with error (abend)</td>
</tr>
</tbody>
</table>
R_LPC_DeepSoftwareStandby

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

[Syntax]

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

[Argument(s)]
None.

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ERROR1</td>
<td>Exit with error (abend)</td>
</tr>
</tbody>
</table>
R_LPC_SoftwareStandby

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ERROR1</td>
<td>Exit with error (abend)</td>
</tr>
</tbody>
</table>
R_LPC_ChangeOperatingPowerControl

Changes the operating power control mode of the MCU.

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

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>operating_mode_t mode;</td>
<td>Operating power control mode type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HIGH_SPEED : High-speed operating mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MIDDLE_SPEED : Middle-speed operating mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LOW_SPEED : Low-speed operating mode</td>
</tr>
</tbody>
</table>

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ERROR1</td>
<td>Change to the low-speed operating mode ended abnormally.</td>
</tr>
<tr>
<td>MD_ERROR2</td>
<td>Change to the middle-speed operating mode is ended abnormally.</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument mode specification</td>
</tr>
</tbody>
</table>
Usage example

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

### 3.2.6 Interrupt controller (ICU)

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

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

Performs initialization necessary to control the interrupt controller.

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_ICU_Create_UserInit

Performs user-defined initialization relating to the interrupt controller.

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

[Syntax]
void R_ICU_Create_UserInit ( void );

[Argument(s)]
None.

[Return value]
None.
r_icu_irqn_interrupt

Performs processing in response to the external pin interrupts.

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

[Syntax]

static void r_icu_irqn_interrupt ( void );

Remark  \( n \) is the source number.

[Argument(s)]

None.

[Return value]

None.
r_icu_software_interrupt

Performs processing in response to the software interrupt.

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

[Syntax]
static void r_icu_software_interrupt ( void );

[Argument(s)]
None.

[Return value]
None.
r_icu_software2_interrupt

Performs processing in response to the software interrupt2.

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_icu_nmi_interrupt

Performs processing in response to the NMI pin interrupt.

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

[Syntax]

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

[Argument(s)]
None.

[Return value]
None.
**R_ICU_IRQn_Start**

Allows detection of the external pin interrupt.

**[Syntax]**

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

**Remark**

$n$ is the source number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_ICU_IRQn_Stop

Prohibits detection of the external pin interrupt.

[Syntax]
void R_ICU_IRQn_Stop ( void );

[Remark]
n is the source number.

[Argument(s)]
None.

[Return value]
None.
**R_ICU_Software_Start**

Allows detection of the software interrupt.

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_ICU_Software2_Start

Allows detection of the software interrupt 2.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
R_ICU_Software_Stop

Prohibits detection of the software interrupt.

[Syntax]
```c
void R_ICU_Software_Stop ( void );
```

[Argument(s)]
None.

[Return value]
None.
**R_ICU_Software2_Stop**

Prohibits detection of the software interrupt 2.

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_ICU_SoftwareInterrupt_Generate

Generates the software interrupt.

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
R_ICU_SoftwareInterrupt2_Generate

Generates the software interrupt 2.

Remark  
\textit{r_icu_software2_interrupt} is called in response to calling od this API function.

[Syntax]

\begin{verbatim}
void R_ICU_SoftwareInterrupt2_Generate ( void );
\end{verbatim}

[Argument(s)]
None.

[Return value]
None.
Usage example

Return from sleep mode by external interrupt.

[GUI setting example]

<table>
<thead>
<tr>
<th>Interrupt Controller Unit</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICU</td>
<td>Used</td>
</tr>
<tr>
<td>IRQ5</td>
<td>Used</td>
</tr>
<tr>
<td>IRQ5</td>
<td>Used</td>
</tr>
<tr>
<td>Pin</td>
<td>P15</td>
</tr>
<tr>
<td>Digital filter</td>
<td>No filter 0(MHz)</td>
</tr>
<tr>
<td>Valid edge</td>
<td>Low level</td>
</tr>
<tr>
<td>Priority</td>
<td>Level 15 (highest)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Low Power Consumption</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPC</td>
<td>Used</td>
</tr>
<tr>
<td>LPC operation setting</td>
<td>Used</td>
</tr>
<tr>
<td>Initial operating power control mode</td>
<td>High-speed operating mode</td>
</tr>
<tr>
<td>Output of the address bus and bus control signals</td>
<td>Retain the output state in software standby mode or deep software standby mode</td>
</tr>
<tr>
<td>Power supply to the on-chip RAM (000A4000h to 000A5FFFh) and USB resumption detecting unit</td>
<td>Power is not supplied in deep software standby mode</td>
</tr>
<tr>
<td>Stop LVD and enable the low power consumption function of the power-on reset circuit</td>
<td>Unused</td>
</tr>
<tr>
<td>Initial sleep mode return clock source</td>
<td>Disabled</td>
</tr>
<tr>
<td>Enable request for release by IRQ0-DS (P30)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable request for release by IRQ1-DS (P31)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable request for release by IRQ2-DS (P32)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable request for release by IRQ3-DS (P33)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable request for release by IRQ4-DS (P34)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable request for release by IRQ5-DS (PA1)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable request for release by IRQ6-DS (PA3)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable request for release by IRQ7-DS (PE2)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable request for release by IRQ8-DS (P40)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable request for release by IRQ9-DS (P41)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable request for release by IRQ10-DS (P42)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable request for release by IRQ11-DS (P43)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable request for release by IRQ12-DS (P44)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable request for release by IRQ13-DS (P45)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable request for release by IRQ14-DS (P46)</td>
<td>Unused</td>
</tr>
<tr>
<td>Feature</td>
<td>Status</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>Enable request for release by IRQ15-DS (P47)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable request for release by NMI</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable request for release by SDA2-DS (P17)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable request for release by SCL2-DS (P16)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable request for release by LVD1</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable request for release by LVD2</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable request for release by CRX1-DS (P15)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable request for release by the RTC periodic interrupt</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable request for release by the RTC alarm interrupt</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable request for release by USB suspension/resumption</td>
<td>Unused</td>
</tr>
<tr>
<td>I/O port state retention</td>
<td>Simultaneous release from deep software standby mode and of retained I/O port state</td>
</tr>
</tbody>
</table>
[API setting example]

```c
r_cg_main.c

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

    /* Enable sleep mode */
    R_LPC_Sleep();

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

```c
r_cg_icu_user.c

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

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

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

Table 3.7 API Functions: [Buses]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_BSC_Create</td>
<td>Performs initialization necessary to control the buses.</td>
</tr>
<tr>
<td>R_BSC_Create_UserInit</td>
<td>Performs user-defined initialization relating to the buses.</td>
</tr>
<tr>
<td>r_bsc_buserrr_interrupt</td>
<td>Performs processing in response to the bus error (illegal address access).</td>
</tr>
<tr>
<td>R_BSC_Error_Monitoring_Start</td>
<td>Allows the detection of bus errors (illegal address access).</td>
</tr>
<tr>
<td>R_BSC_Error_Monitoring_Stop</td>
<td>Prohibits the detection of bus errors (illegal address access).</td>
</tr>
<tr>
<td>R_BSC_InitializeSDRAM</td>
<td>Performs initialization of SDRAM controller.</td>
</tr>
</tbody>
</table>
**R_BSC_Create**

Performs initialization necessary to control the buses.

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_BSC_Create_UserInit

Performs user-defined initialization relating to the buses.

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

[Syntax]

\[
\textbf{void} \quad \text{R\_BSC\_Create\_UserInit ( void )};
\]

[Argument(s)]

None.

[Return value]

None.
r_bsc_buserr_interrupt

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

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

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

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

[Syntax]

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

[Argument(s)]
None.

[Return value]
None.
**R_BSC_Error_Monitoring_Start**

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_BSC_Error_Monitoring_Stop**

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_BSC_InitializeSDRAM

Performs initialization of SDRAM controller.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
Usage example

Get the address where bus error occurred.

[GUI setting example]

<table>
<thead>
<tr>
<th>Buses</th>
<th>Used</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BSC0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bus operation setting</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>On-Chip ROM setting</td>
<td>Enable</td>
<td></td>
</tr>
<tr>
<td>ALE pin</td>
<td>Unused</td>
<td></td>
</tr>
<tr>
<td>WAIT# pin</td>
<td>Unused</td>
<td></td>
</tr>
<tr>
<td>BC1#/WR1# pin</td>
<td>Unused</td>
<td></td>
</tr>
<tr>
<td>Memory bus 1 and 3 (RAM/Extended RAM)</td>
<td>The order of priority is fixed</td>
<td></td>
</tr>
<tr>
<td>0000 0000h to 0007 FFFFh</td>
<td>0080 0000h to 00FF FFFFh</td>
<td></td>
</tr>
<tr>
<td>Memory bus 2 (ROM)</td>
<td>The order of priority is fixed</td>
<td></td>
</tr>
<tr>
<td>8000 0000h to FEFF FFFFh</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internal peripheral bus 1 (Peripheral I/O registers)</td>
<td>The order of priority is fixed</td>
<td></td>
</tr>
<tr>
<td>0008 0000h to 0008 7FFFh</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internal peripheral bus 2 and 3 (Peripheral I/O registers)</td>
<td>The order of priority is fixed</td>
<td></td>
</tr>
<tr>
<td>0008 8000h to 0009 FFFFh 000A 0000h to 000B FFFFh</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internal peripheral bus 4 and 5 (Peripheral I/O registers)</td>
<td>The order of priority is fixed</td>
<td></td>
</tr>
<tr>
<td>000C 0000h to 000D FFFFh 000E 0000h to 000F FFFFh</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internal peripheral bus 6 (E2 DataFlash memory and ROM)</td>
<td>The order of priority is fixed</td>
<td></td>
</tr>
<tr>
<td>0010 0000h to 00FF FFFFh</td>
<td></td>
<td></td>
</tr>
<tr>
<td>External bus 0100 0000h to 0FFF FFFFh</td>
<td>The order of priority is fixed</td>
<td></td>
</tr>
<tr>
<td>Separate bus CS recovery cycle insertion enable setting Read access after read access (Same area)</td>
<td>Unused</td>
<td></td>
</tr>
<tr>
<td>Separate bus CS recovery cycle insertion enable setting Read access after read access (Different area)</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>Separate bus CS recovery cycle insertion enable setting Write access after read access (Same area)</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>Separate bus CS recovery cycle insertion enable setting Write access after read access (Different area)</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>Separate bus CS recovery cycle insertion enable setting Read access after write access (Same area)</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>Code Generator Tool 3.API FUNCTIONS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Separate bus CS recovery cycle insertion enable setting Read access after write access (Different area)</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>Separate bus CS recovery cycle insertion enable setting Write access after write access (Same area)</td>
<td>Unused</td>
<td></td>
</tr>
<tr>
<td>Separate bus CS recovery cycle insertion enable setting Write access after write access (Different area)</td>
<td>Unused</td>
<td></td>
</tr>
<tr>
<td>Address/data multiplexed bus CS recovery cycle insertion enable setting Read access after read access (Same area)</td>
<td>Unused</td>
<td></td>
</tr>
<tr>
<td>Address/data multiplexed bus CS recovery cycle insertion enable setting Read access after read access (Different area)</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>Address/data multiplexed bus CS recovery cycle insertion enable setting Write access after read access (Same area)</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>Address/data multiplexed bus CS recovery cycle insertion enable setting Write access after read access (Different area)</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>Address/data multiplexed bus CS recovery cycle insertion enable setting Read access after write access (Same area)</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>Address/data multiplexed bus CS recovery cycle insertion enable setting Read access after write access (Different area)</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>Address/data multiplexed bus CS recovery cycle insertion enable setting Write access after write access (Same area)</td>
<td>Unused</td>
<td></td>
</tr>
<tr>
<td>Address/data multiplexed bus CS recovery cycle insertion enable setting Write access after write access (Different area)</td>
<td>Unused</td>
<td></td>
</tr>
<tr>
<td>A7-A0, BC0#, DQM2, DQM3 (PA7-PA0)</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>A8</td>
<td>PB0</td>
<td></td>
</tr>
<tr>
<td>A9</td>
<td>PB1</td>
<td></td>
</tr>
<tr>
<td>A10</td>
<td>PB2</td>
<td></td>
</tr>
<tr>
<td>A11</td>
<td>PB3</td>
<td></td>
</tr>
<tr>
<td>A12</td>
<td>PB4</td>
<td></td>
</tr>
<tr>
<td>A13</td>
<td>PB5</td>
<td></td>
</tr>
<tr>
<td>A14</td>
<td>PB6</td>
<td></td>
</tr>
<tr>
<td>A15</td>
<td>PB7</td>
<td></td>
</tr>
</tbody>
</table>
### 3. API FUNCTIONS

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

r_cg_main.c

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

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

r_cg_bsc_user.c

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

static void r_bsc_buserr_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Restore an address that was accessed when a bus error occurred */
    if (1U == BSC.BERSR1.BIT.IA)
    {
        g_bsc_buserr_addr = ((uint16_t)(BSC.BERSR2.WORD)>>3U);
    }
    /* Clear the bus error status registers */
    BSC.BERCLR.BIT.STSCLR = 1U;
    /* End user code. Do not edit comment generated here */
}
```
3.2.8 DMA Controller (DMAC)

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

Table 3.8 API Functions: [DMA Controller]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_DMAC_Create</td>
<td>Performs initialization necessary to control the DMA controller.</td>
</tr>
<tr>
<td>R_DMACn_Start</td>
<td>Allows starting of the DMAC controller.</td>
</tr>
<tr>
<td>R_DMACn_Stop</td>
<td>Prohibits starting of the DMAC controller.</td>
</tr>
<tr>
<td>R_DMACn_Set_SoftwareTrigger</td>
<td>Sets the software request of DMA transfer by software.</td>
</tr>
<tr>
<td>R_DMACn_Clear_SoftwareTrigger</td>
<td>Clears the software request of DMA transfer by software.</td>
</tr>
<tr>
<td>r_dmac_dmacni_interrupt</td>
<td>Performs processing in response to the transfer end interrupt.</td>
</tr>
<tr>
<td>r_dmacn_callback_transfer_end</td>
<td>Performs processing in response to the transfer end interrupt.</td>
</tr>
<tr>
<td>r_dmacn_callback_transfer_escape_end</td>
<td>Performs processing in response to the escape transfer end interrupt.</td>
</tr>
<tr>
<td>R_DMAC_Create_UserInit</td>
<td>Performs user-defined initialization relating to the DMA controller.</td>
</tr>
</tbody>
</table>
**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]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_DMACh_Start

Allows starting of the DMAC controller.

**[Syntax]**

```c
void R_DMACh_Start ( void );
```

**Remark**

\( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_DMACn_Stop

Prohibits starting of the DMAC controller.

[Syntax]

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

Remark  

$n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
**R_DMACn_Set_SoftwareTrigger**

Sets the software request of DMA transfer by software.

**[Syntax]**

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

**Remark**

$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_DMACn_Clear_SoftwareTrigger

Clears the software request of DMA transfer by software.

[Syntax]

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

Remark  

$n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
**r_dmac_dmacni_interrupt**

Performs processing in response to the transfer end interrupt.

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

**[Syntax]**

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

**Remark**
$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_dmacn_callback_transfer_end

Performs processing in response to the transfer end interrupt.

Remark This API function is called as the call r_dmac_dmacni_interrupt back routine.

[Syntax]

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

Remark \(n\) is the channel number.

[Argument(s)]

None.

[Return value]

None.
**r_dmacn_callback_transfer_escape_end**

Performs processing in response to the escape transfer end interrupt.

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

**[Syntax]**

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

**Remark**
`n` is the channel number.

**[Argument(s)]**
None.

**[Return value]**
None.
R_DMAC_Create_UserInit

Performs user-defined initialization relating to the DMA controller.

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

[Syntax]

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

[Argument(s)]
None.

[Return value]
None.
**Usage example**

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

[GUI setting example]

<table>
<thead>
<tr>
<th>DMA Controller</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dmac</td>
<td>Used</td>
</tr>
<tr>
<td>DmacChannel0</td>
<td>Used</td>
</tr>
<tr>
<td>Activation source</td>
<td>CMT0 (CMI0 vect:28)</td>
</tr>
<tr>
<td>Activation source flag control</td>
<td>Clear interrupt flag of the activation source</td>
</tr>
<tr>
<td>Transfer mode</td>
<td>Normal mode</td>
</tr>
<tr>
<td>Transfer data size</td>
<td>8 bits</td>
</tr>
<tr>
<td>Transfer count</td>
<td>1</td>
</tr>
<tr>
<td>Total transfer size</td>
<td>1byte(s)</td>
</tr>
<tr>
<td>Source address</td>
<td>0x00000100 (Fixed)</td>
</tr>
<tr>
<td>Destination address</td>
<td>0x00000110 (Fixed)</td>
</tr>
<tr>
<td>Interrupt setting (DMAC0I)</td>
<td>Used</td>
</tr>
<tr>
<td>Enable interrupt on transfer end</td>
<td>Used</td>
</tr>
<tr>
<td>Priority</td>
<td>Level 15 (highest)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compare Match Timer</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMT0</td>
<td>Used</td>
</tr>
<tr>
<td>Compare match timer operation setting</td>
<td>Used</td>
</tr>
<tr>
<td>Count clock setting</td>
<td>PCLK/32</td>
</tr>
<tr>
<td>Interval value setting</td>
<td>100ms (Actual value : 100)</td>
</tr>
<tr>
<td>Enable compare match interrupt (CMI0)</td>
<td>Used</td>
</tr>
<tr>
<td>Priority</td>
<td>Level 15 (highest)</td>
</tr>
</tbody>
</table>
[API setting example]

r_cg_main.c

```c
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start CMT channel 0 counter */
    R_CMT0_Start();
    /* Enable the DMAC0 activation */
    R_DMAC0_Start();

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

r_cg_dmac_user.c

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

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

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

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

Table 3.9 API Functions: [Data Transfer Controller]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_DTC_Create</td>
<td>Performs initialization necessary to control the data transfer controller.</td>
</tr>
<tr>
<td>R_DTC_Create_UserInit</td>
<td>Performs user-defined initialization relating to the data transfer controller.</td>
</tr>
<tr>
<td>R_DTCm_Start</td>
<td>Allows starting of the data transfer controller.</td>
</tr>
<tr>
<td>R_DTCm_Stop</td>
<td>Prohibits starting of the data transfer controller.</td>
</tr>
</tbody>
</table>
R_DTC_Create

Performs initialization necessary to control the data transfer controller.
Remark  This API function is called from R_Systeminit before main() is executed.

[Syntax]

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

[Argument(s)]
None.

[Return value]
None.
**R_DTC_Create_UserInit**

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

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_DTCm_Start**

Allows starting the data transfer controller.

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

$m$ is the transfer data number, $n$ is the interrupt vector number.

**[Syntax]**

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

**Remark**  
$m$ is the transfer data number.

**[Argument(s)]**

None.

**[Return value]**

None.
### R_DTCm_Stop

Prohibits starting of the data transfer controller.

**Remark**

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

$m$ is the transfer data number, $n$ is the interrupt vector number.

**[Syntax]**

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

**Remark**

$m$ is the transfer data number.

**[Argument(s)]**

None.

**[Return value]**

None.
Usage example

The transfer starts by compare match interrupt.

[GUI setting example]

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

<table>
<thead>
<tr>
<th>Compare Match Timer</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMT0</td>
<td>Used</td>
</tr>
<tr>
<td>Compare match timer</td>
<td>Used</td>
</tr>
<tr>
<td>operation setting</td>
<td>Used</td>
</tr>
<tr>
<td>Count clock setting</td>
<td>PCLK/32</td>
</tr>
<tr>
<td>Interval value setting</td>
<td>100ms (Actual value : 100)</td>
</tr>
<tr>
<td>Enable compare match</td>
<td>Used</td>
</tr>
<tr>
<td>interrupt (CMI0)</td>
<td></td>
</tr>
<tr>
<td>Priority</td>
<td>Level 15 (highest)</td>
</tr>
</tbody>
</table>
[API setting example]

r_cg_main.c

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

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

    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```
### 3.2.10 Event link controller (ELC)

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_ELC_Create</td>
<td>Performs initialization necessary to control the event link controller.</td>
</tr>
<tr>
<td>R_ELC_Create_UserInit</td>
<td>Performs user-defined initialization relating to the event link controller.</td>
</tr>
<tr>
<td>r_elc_ei_{int}</td>
<td>Performs processing in response to the event link interrupt.</td>
</tr>
<tr>
<td>R_ELC_Start</td>
<td>Starts interlinked operation of peripheral functions.</td>
</tr>
<tr>
<td>R_ELC_Stop</td>
<td>Ends interlinked operation of peripheral functions.</td>
</tr>
<tr>
<td>R_ELC_GenerateSoftwareEvent</td>
<td>Generates the software event.</td>
</tr>
<tr>
<td>R_ELC_Set_PortBuffern</td>
<td>Sets the value of a port buffer.</td>
</tr>
<tr>
<td>R_ELC_Get_PortBuffern</td>
<td>Gets the value of a port buffer.</td>
</tr>
</tbody>
</table>
R_ELC_Create

Performs initialization necessary to control the event link controller.

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
R_ELC_Create_UserInit

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

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
r_elc_elsrni_interrupt

Performs processing in response to the event link interrupt.

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

is the source number.

[Syntax]

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

Remark n is the source number.

[Argument(s)]

None.

[Return value]

None.
R_ELC_Start

Starts interlinked operation of peripheral functions.

[Syntax]

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

[Argument(s)]
None.

[Return value]
None.
R_ELC_Stop

Ends interlinked operation of peripheral functions.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
**R_ELC_GenerateSoftwareEvent**

Generates the software event.

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_ELC_Set_PortBuffer n

Sets the value of a port buffer.

[Syntax]

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

Remark  

\( n \) is the source number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint8_t value;</td>
<td>The value set in the port buffer.</td>
</tr>
</tbody>
</table>

[Return value]

None.
**R_ELC_Get_PortBuffer**

Gets the value of a port buffer.

**Syntax**

```c
void R_ELC_Get_PortBuffer ( uint8_t * const value );
```

Remark  

$n$ is the source number.

**Argument(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint8_t * const value;</td>
<td>Pointer to the location where the obtained value is to be stored.</td>
</tr>
</tbody>
</table>

**Return value**

None.
Usage example

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

[GUI setting example]

<table>
<thead>
<tr>
<th>Event Link Controller</th>
<th>ELC</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELC_C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MT1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMT1</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>Event signal</td>
<td></td>
<td>Software event</td>
</tr>
<tr>
<td>Operation on event</td>
<td></td>
<td>Counting is started</td>
</tr>
<tr>
<td>ELC Interrupt1</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>Interrupt 1</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>Event signal</td>
<td></td>
<td>Input edge detection of input port group 2</td>
</tr>
<tr>
<td>Operation on event</td>
<td></td>
<td>Issues an event to the CPU</td>
</tr>
<tr>
<td>ELSR18I Priority</td>
<td></td>
<td>Level 15 (highest)</td>
</tr>
<tr>
<td>ELC Interrupt2</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>Interrupt 2</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>Event signal</td>
<td></td>
<td>Input edge detection of input port group 2</td>
</tr>
<tr>
<td>Operation on event</td>
<td></td>
<td>Issues an event to the CPU</td>
</tr>
<tr>
<td>ELSR19I Priority</td>
<td></td>
<td>Level 15 (highest)</td>
</tr>
<tr>
<td>Port Group 2</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>Port group 2 setting</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>PE0</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>PE1</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>PE2</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>PE3</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>PE4</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>PE5</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>PE6</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>PE7</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>Input port group 2 setting</td>
<td>Used</td>
<td>Input edge for event generation</td>
</tr>
<tr>
<td>Enable PDBF2 overwrite</td>
<td>Unused</td>
<td></td>
</tr>
<tr>
<td>Input port group 2 setting</td>
<td>CMT1 compare match 1</td>
<td></td>
</tr>
<tr>
<td>Event signal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input port group 2 setting</td>
<td>Transfers the signal value of the external pin to the PDBFn register</td>
<td></td>
</tr>
<tr>
<td>Operation on event</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Compare Match Timer

<table>
<thead>
<tr>
<th>Compare Match Timer</th>
<th>CMT1</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Used</td>
</tr>
<tr>
<td>Compare match timer operation setting</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>Count clock setting</td>
<td>PCLK/512</td>
<td></td>
</tr>
<tr>
<td>Interval value setting</td>
<td>3000ms (Actual value : 2999.978667)</td>
<td></td>
</tr>
<tr>
<td>Enable compare match interrupt (CMT1)</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>Priority</td>
<td>Level 15 (highest)</td>
<td></td>
</tr>
</tbody>
</table>
API setting example

r_cg_main.c

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

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

r_cg_elc_user.c

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

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

Table 3.11 API Functions: [I/O Ports]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_PORT_Create</td>
<td>Performs initialization necessary to control the I/O ports.</td>
</tr>
<tr>
<td>R_PORT_Create_UserInit</td>
<td>Performs user-defined initialization relating to the I/O ports.</td>
</tr>
</tbody>
</table>
R_PORT_Create

Performs initialization necessary to control the I/O ports.

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

[Syntax]

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

[Argument(s)]
None.

[Return value]
None.
**R_PORT_Create_UserInit**

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

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
### 3.2.12 Multi-function timer pulse unit 2 (MTU2)

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

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_MTU2_Create</td>
<td>Performs initialization necessary to control the multi-function timer pulse unit 2.</td>
</tr>
<tr>
<td>R_MTU2_Create_UserInit</td>
<td>Performs user-defined initialization relating to the multi-function timer pulse unit 2.</td>
</tr>
<tr>
<td>r_mtu2_tgimn_interrupt</td>
<td>Performs processing in response to the input capture/compare match interrupt.</td>
</tr>
<tr>
<td>r_mtu2_cj_tgimn_interrupt</td>
<td>Performs processing in response to the input capture/compare match interrupt.</td>
</tr>
<tr>
<td>r_mtu2_tciun_interrupt</td>
<td>Performs processing in response to the overflow interrupt.</td>
</tr>
<tr>
<td>r_mtu2_cj_tciun_interrupt</td>
<td>Performs processing in response to the overflow interrupt.</td>
</tr>
<tr>
<td>R_MTU2_Cn_Start</td>
<td>Starts counting by a 16-bit timer.</td>
</tr>
<tr>
<td>R_MTU2_Cn_Stop</td>
<td>Ends counting by a 16-bit timer.</td>
</tr>
</tbody>
</table>
R_MTU2_Create

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

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
R_MTU2_Create_UserInit

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

Remark

This API function is called as the R_MTU2_Create callback routine.

[Syntax]

void R_MTU2_Create_UserInit ( void );

[Argument(s)]

None.

[Return value]

None.
**r_mtu2_tgimn_interrupt**

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

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

**Syntax**

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

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

**Argument(s)**

None.

**Return value**

None.
r_mtu2_cj_tgimn_interrupt

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

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

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.
r_mtu2_tcivn_interrupt

Performs processing in response to the overflow interrupt.

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

[Syntax]

    static void r_mtu2_tcivn_interrupt ( void );

Remark     n is the channel number.

[Argument(s)]
None.

[Return value]  
None.
**r_mtu2_cj_tcivn_interrupt**

Performs processing in response to the overflow interrupt.

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

**[Syntax]**

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

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_mtu2_tciun_interrupt

Performs processing in response to the underflow interrupt.

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

[Syntax]

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

Remark $n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_MTU2_Cn_Start

Starts counting by the 16-bit timer.

[Syntax]

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

Remark $n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_MTU2_Cn_Stop

Ends counting by the 16-bit timer.

[Syntax]

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

Remark  

$n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
Usage example

Use timer as One-shot timer.

[GUI setting example]

<table>
<thead>
<tr>
<th>Multi-Function Timer Pulse Unit</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>MTU2_U 0</td>
<td></td>
</tr>
<tr>
<td>MTU0</td>
<td>Used</td>
</tr>
<tr>
<td>MTU0</td>
<td>Normal mode</td>
</tr>
<tr>
<td>Include this channel in the synchronous operation</td>
<td>Unused</td>
</tr>
<tr>
<td>Counter clock selection</td>
<td>PCLK</td>
</tr>
<tr>
<td>Counter clear source</td>
<td>TGRA0 compare match/input capture (Use TGRA0 as a cycle register)</td>
</tr>
<tr>
<td>TGRA0 (Output compare register)</td>
<td>50ms (Actual value : 50)</td>
</tr>
<tr>
<td>TGRB0 (Output compare register)</td>
<td>100μs (Actual value : 100)</td>
</tr>
<tr>
<td>TGRC0 (Output compare register)</td>
<td>100μs (Actual value : 100)</td>
</tr>
<tr>
<td>TGRD0 (Output compare register)</td>
<td>100μs (Actual value : 100)</td>
</tr>
<tr>
<td>TGRE0 (Output compare register)</td>
<td>100μs (Actual value : 100)</td>
</tr>
<tr>
<td>TGRF0 (Output compare register)</td>
<td>100μs (Actual value : 100)</td>
</tr>
<tr>
<td>MTIOC0A pin (PB3)</td>
<td>MTIOC0A pin output disabled</td>
</tr>
<tr>
<td>MTIOC0B pin (P13)</td>
<td>MTIOC0B pin output disabled</td>
</tr>
<tr>
<td>MTIOC0C pin (P32)</td>
<td>MTIOC0C pin output disabled</td>
</tr>
<tr>
<td>MTIOC0D pin (P33)</td>
<td>MTIOC0D pin output disabled</td>
</tr>
<tr>
<td>Enable A/D conversion start request on TGRA input capture/compare match (trigger signal of MTU0 TGRA0N)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable TGRA0 input capture/compare match interrupt (TGIA0)</td>
<td>Used</td>
</tr>
<tr>
<td>Enable TGRB0 input capture/compare match interrupt (TGIB0)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable TGRC0 input capture/compare match interrupt (TGIC0)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable TGRD0 input capture/compare match interrupt (TGID0)</td>
<td>Unused</td>
</tr>
<tr>
<td>(TGIA/TGIB/TGIC/TGID) Priority</td>
<td>Level 15 (highest)</td>
</tr>
<tr>
<td>Enable TGRE0 compare match interrupt (TGIE0)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable TGRF0 compare match interrupt (TGIF0)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable overflow interrupt (TCIV0)</td>
<td>Unused</td>
</tr>
</tbody>
</table>
[API setting example]

r_cg_main.c

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

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

r_cg_mtu2_user.c

```c
static void r_mtu2_tgia0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop MTU2 channel 0 counter */
    R_MTU2_C0_Stop();
    /* End user code. Do not edit comment generated here */
}
```
3.2.13 Multi-function timer pulse unit 3 (MTU3)

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

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

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

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

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_MTU3_Create_UserInit

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

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

[Syntax]

void R_MTU3_Create_UserInit ( void );

[Argument(s)]

None.

[Return value]

None.
r_mtu3_tgimn_interrupt

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

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

[Syntax]

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

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

[Argument(s)]
None.

[Return value]
None.
r_mtu3_cj_tgimn_interrupt

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

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

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.
### r_mtu3_tcivn_interrupt

Performs processing in response to the overflow interrupt.

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

**Syntax**

```c
static void r_mtu3_tcivn_interrupt ( void );
```

**Remark** \( n \) is the channel number.

**Argument(s)**

None.

**Return value**

None.
**r_mtu3_cj_tcivn_interrupt**

Performs processing in response to the overflow interrupt.

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

**[Syntax]**

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

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_mtu3_tciun_interrupt**

Performs processing in response to the underflow interrupt.

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

**[Syntax]**

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

**Remark** \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_MTU3_Cn_Start**

Starts counting by the 16-bit timer.

**[Syntax]**

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

Remark  
$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_MTU3_Cn_Stop**

Ends counting by the 16-bit timer.

**[Syntax]**

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

**Remark**

$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
### Usage example

Use timer as One-shot timer.

#### [GUI setting example]

<table>
<thead>
<tr>
<th>Multi-Function Timer Pulse Unit 3</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTU3_U</td>
<td>Used</td>
</tr>
<tr>
<td>MTCLKA pin</td>
<td>Unused</td>
</tr>
<tr>
<td>MTCLKB pin</td>
<td>Unused</td>
</tr>
<tr>
<td>MTCLKC pin</td>
<td>Unused</td>
</tr>
<tr>
<td>MTCLKD pin</td>
<td>Unused</td>
</tr>
<tr>
<td>MTU0</td>
<td>Used</td>
</tr>
<tr>
<td>MTU0</td>
<td>Normal mode</td>
</tr>
<tr>
<td>Include this channel in the synchronous operation</td>
<td>Unused</td>
</tr>
<tr>
<td>Counter clock selection</td>
<td>PCLK/64</td>
</tr>
<tr>
<td>Counter clear source</td>
<td>TGRA0 compare match/input capture (Use TGRA0 as a cycle register)</td>
</tr>
<tr>
<td>TGRA0 (Output compare register)</td>
<td>50ms (Actual value: 50.005333)</td>
</tr>
<tr>
<td>TGRB0 (Output compare register)</td>
<td>100μs (Actual value: 96)</td>
</tr>
<tr>
<td>TGRC0 (Output compare register)</td>
<td>100μs (Actual value: 96)</td>
</tr>
<tr>
<td>TGRD0 (Output compare register)</td>
<td>100μs (Actual value: 96)</td>
</tr>
<tr>
<td>TGRE0 (Output compare register)</td>
<td>100μs (Actual value: 96)</td>
</tr>
<tr>
<td>TGRF0 (Output compare register)</td>
<td>100μs (Actual value: 96)</td>
</tr>
<tr>
<td>MTIOC0A pin (P34)</td>
<td>MTIOC0A pin output is disabled</td>
</tr>
<tr>
<td>MTIOC0B pin (P13)</td>
<td>MTIOC0B pin output is disabled</td>
</tr>
<tr>
<td>MTIOC0C pin (P32)</td>
<td>MTIOC0C pin output is disabled</td>
</tr>
<tr>
<td>MTIOC0D pin (P33)</td>
<td>MTIOC0D pin output is disabled</td>
</tr>
<tr>
<td>Enable A/D conversion start request on TGRA input capture/compare match (trigger signal of MTU0 TRGA0N)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable A/D conversion start request on TGRE compare match (trigger signal of MTU0 TRG0N)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable TGRA0 input capture/compare match interrupt (TGIA0)</td>
<td>Level 15 (highest)</td>
</tr>
<tr>
<td>Enable TGRB0 input capture/compare match interrupt (TGIB0)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable TGRC0 input capture/compare match interrupt (TGIC0)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable TGRD0 input capture/compare match interrupt (TGID0)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable TGRE0 compare match interrupt (TGIE0)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable TGRF0 compare</td>
<td>Unused</td>
</tr>
</tbody>
</table>
match interrupt (TGIF0)
Enable overflow interrupt (TCIV0)
Unused

[API setting example]

r_cg_main.c

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

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

r_cg_mtu3_user.c

```c
static void r_mtu3_tgia0_interrupt(void)
{
  /* Start user code. Do not edit comment generated here */
  /* Stop MTU3 channel 0 counter */
  R_MTU3_C0_Stop();
  /* End user code. Do not edit comment generated here */
}
```
### 3.2.14 Port output enable 2 (POE2)

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_POE2_Create</td>
<td>Performs initialization necessary to control the port output enable 2.</td>
</tr>
<tr>
<td>R_POE2_Create_UserInit</td>
<td>Performs user-defined initialization relating to the port output enable 2.</td>
</tr>
<tr>
<td>r_poe2_oein_interrupt</td>
<td>Performs processing in response to the output enable interrupt n (OEIn).</td>
</tr>
<tr>
<td>R_POE2_Start</td>
<td>Places the MTU's complementary PWM output pins in the high-impedance state.</td>
</tr>
<tr>
<td>R_POE2_Stop</td>
<td>Releases the R_POE2_Stop MTU's complementary PWM output pins from the high-impedance state.</td>
</tr>
<tr>
<td>R_POE2_Set_HiZ_MTUn</td>
<td>Sets the high-impedance state for the MTUn pins.</td>
</tr>
<tr>
<td>R_POE2_Clear_HiZ_MTUn</td>
<td>Clear the high-impedance state for the MTUn pins.</td>
</tr>
</tbody>
</table>
R_POE2_Create

Performs initialization necessary to control the port output enable 2.

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
R_POE2_Create_UserInit

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

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

Syntax

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

Argument(s)

None.

Return value

None.
r_poe2_oein_interrupt

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

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

[Syntax]
static void r_poe2_oein_interrupt ( void );

Remark n is the channel number.

[Argument(s)]
None.

[Return value]
None.
R_POE2_Start

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
R_POE2_Stop

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

[Syntax]

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

[Argument(s)]
None.

[Return value]
None.
R_POE2_Set_HiZ_MTUn

Sets the high-impedance state for the MTUn pins.

**[Syntax]**

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

Remark \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_POE2_Clear_HiZ_MTUn

Clear the high-impedance state for the MTUn pins.

[Syntax]

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

Remark  

\( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
Usage example

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

[GUI setting example]

<table>
<thead>
<tr>
<th>Port Output Enable 2</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>POE2</td>
<td>Used</td>
</tr>
<tr>
<td>Enable MTIOC0A</td>
<td>Used</td>
</tr>
<tr>
<td>Enable MTIOC0B</td>
<td>Used</td>
</tr>
<tr>
<td>Enable MTIOC0C</td>
<td>Used</td>
</tr>
<tr>
<td>Enable MTIOC0D</td>
<td>Used</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>POE8# pin</th>
<th>P17</th>
</tr>
</thead>
<tbody>
<tr>
<td>POE8# mode select</td>
<td>On the falling edge of POE8# input</td>
</tr>
<tr>
<td>Output enable interrupt 2 (OEI2)</td>
<td>Used</td>
</tr>
<tr>
<td>OEI2 Priority</td>
<td>Level 15 (highest)</td>
</tr>
<tr>
<td>Enable MTIOC4A and MTIOC4C</td>
<td>MTIOC4C</td>
</tr>
<tr>
<td>Enable MTIOC4B and MTIOC4D</td>
<td>MTIOC4D</td>
</tr>
<tr>
<td>Output enable interrupt 1 (OEI1)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable request to place pins in the high-impedance on detection of stopped oscillation</td>
<td>Unused</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Multi-Function Timer Pulse Unit 2</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTU2_U</td>
<td>Used</td>
</tr>
<tr>
<td>MTU0</td>
<td>Used</td>
</tr>
<tr>
<td>MTU0</td>
<td>Used</td>
</tr>
<tr>
<td>Include this channel in the synchronous operation</td>
<td>Unused</td>
</tr>
<tr>
<td>Counter clock selection</td>
<td>PCLK</td>
</tr>
<tr>
<td>Counter clear source</td>
<td>Disabled counter clear</td>
</tr>
<tr>
<td>TGRD0</td>
<td>Output compare register</td>
</tr>
<tr>
<td>TGRD0</td>
<td>Output compare register</td>
</tr>
<tr>
<td>TGRE0</td>
<td>Output compare register</td>
</tr>
<tr>
<td>TGRF0</td>
<td>Output compare register</td>
</tr>
</tbody>
</table>

| MTIOC0A pin (Initial output of MTIOC0A pin is 0. 0 output at compare match.) | P34 |
| When TGRB compare match | 0 output from MTIOC0A pin |
| MTIOC0C pin (Initial output of MTIOC0C pin is 0. 0 output at compare match.) | P32 |
| When TGRD compare match | 0 output from MTIOC0C pin |

Initial of compare match A value (TGRA) 100
Initial of compare match B value (TGRB) 100
Initial of compare match C 100
<table>
<thead>
<tr>
<th>Function Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial of compare match D value (TGRD)</td>
<td>100</td>
</tr>
<tr>
<td>Initial of compare match E value (TGRE)</td>
<td>100</td>
</tr>
<tr>
<td>Initial of compare match F value (TGRF)</td>
<td>100</td>
</tr>
<tr>
<td>Enable A/D conversion start request on TGRA input capture/compare match (trigger signal of MTU0 TRGA0N)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable TGRA0 input capture/compare match interrupt (TGIA0)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable TGRB0 input capture/compare match interrupt (TGIB0)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable TGRC0 input capture/compare match interrupt (TGIC0)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable TGRD0 input capture/compare match interrupt (TGID0)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable TGRE0 compare match interrupt (TGIE0)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable TGRF0 compare match interrupt (TGIF0)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable overflow interrupt (TCIV0)</td>
<td>Unused</td>
</tr>
</tbody>
</table>
[API setting example]

**r_cg_main.c**

```c
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start the POE2 module */
    R_POE2_Start();
    
    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

**r_cg_poe2_user.c**

```c
static void r_poe2_oei2_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop the POE2 module */
    R_POE2_Stop();
    /* End user code. Do not edit comment generated here */
}
```
## 3.2.15 Port output enable 3 (POE3)

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

### Table 3.15 API Functions: [Port Output Enable 3]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_POE3_Create</td>
<td>Performs initialization necessary to control the port output enable 3.</td>
</tr>
<tr>
<td>R_POE3_Create_UserInit</td>
<td>Performs user-defined initialization relating to the port output enable 3.</td>
</tr>
<tr>
<td>r_poe3_oein_interrupt</td>
<td>Performs processing in response to the output enable interrupt n (OEIn).</td>
</tr>
<tr>
<td>R_POE3_Start</td>
<td>Places the MTU's complementary PWM output pins in the high-impedance state.</td>
</tr>
<tr>
<td>R_POE3_Stop</td>
<td>Releases the R_POE3_Stop MTU's complementary PWM output pins from the high-impedance state.</td>
</tr>
<tr>
<td>R_POE3_Set_HiZ_MTUn</td>
<td>Sets the high-impedance state for the MTUn pins.</td>
</tr>
<tr>
<td>R_POE3_Clear_HiZ_MTUn</td>
<td>Clear the high-impedance state for the MTUn pins.</td>
</tr>
<tr>
<td>R_POE3_Set_HiZ_GPTn</td>
<td>Sets the high-impedance state for the GPTn pins.</td>
</tr>
<tr>
<td>R_POE3_Clear_HiZ_GPTn</td>
<td>Clear the high-impedance state for the GPTn pins.</td>
</tr>
</tbody>
</table>
R_POE3_Create

Performs initialization necessary to control the port output enable 3.

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

.Syntax

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

[Argument(s)]

None.

[Return value]

None.
R_POE3_Create_UserInit

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

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
Perform processing in response to the output enable interrupt.

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

**[Syntax]**

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

**Remark**
$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_POE3_Start**

Places the related pins in the high-impedance state.

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_POE3_Stop

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

[Syntax]

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

[Argument(s)]
None.

[Return value]
None.
R_POE3_Set_HiZ_MTUn

Sets the high-impedance state for the MTUn pins.

**[Syntax]**

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

Remark  

$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_POE3_Clear_HiZ_MTUn

Clear the high-impedance state for the MTUn pins.

[Syntax]

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

Remark  

$n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_POE3_Set_HiZ_GPTn

Sets the high-impedance state for the GPTn pins.

[Syntax]
void R_POE3_Set_HiZ_GPTn ( void );

Remark n is the channel number.

[Argument(s)]
None.

[Return value]
None.
R_POE3_Clear_HiZ_GPTn

Clear the high-impedance state for the GPTn pins.

**[Syntax]**

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

Remark  

$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
Usage example

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

[GUI setting example]

<table>
<thead>
<tr>
<th>Port Output Enable 3</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>POE3</td>
<td>Used</td>
</tr>
<tr>
<td>MTU0 output pin control setting</td>
<td></td>
</tr>
<tr>
<td>Enable MTIOC0A</td>
<td>Used</td>
</tr>
<tr>
<td>MTIOC0A pin</td>
<td>Controls high-impedance state of P34</td>
</tr>
<tr>
<td>Enable MTIOC0B</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable MTIOC0C</td>
<td>Used</td>
</tr>
<tr>
<td>MTIOC0C pin</td>
<td>Controls high-impedance state of P32</td>
</tr>
<tr>
<td>Enable MTIOC0D</td>
<td>Unused</td>
</tr>
<tr>
<td>POE0# input level detection (MTU0)</td>
<td>Unused</td>
</tr>
<tr>
<td>POE4# input level detection (MTU0)</td>
<td>Unused</td>
</tr>
<tr>
<td>POE10# input level detection (MTU0)</td>
<td>Unused</td>
</tr>
<tr>
<td>POE11# input level detection (MTU0)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable POE8 input</td>
<td>Used</td>
</tr>
<tr>
<td>POE8# pin</td>
<td>P17</td>
</tr>
<tr>
<td>POE8 mode select</td>
<td>On the falling edge of POE8# input</td>
</tr>
<tr>
<td>Output enable interrupt 3 (OEI3)</td>
<td>Used</td>
</tr>
<tr>
<td>OEI3 Priority (Group BL1)</td>
<td>Level 15 (highest)</td>
</tr>
<tr>
<td>MTU3 and MTU4/GPT0 GPT1 and GPT2 output pin control setting</td>
<td></td>
</tr>
<tr>
<td>MTU3/GPT0 select</td>
<td>Unused</td>
</tr>
<tr>
<td>MTU4/GPT1 select</td>
<td>Unused</td>
</tr>
<tr>
<td>MTU4/GPT2 select</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable POE0 input</td>
<td>Unused</td>
</tr>
<tr>
<td>Output enable interrupt 1 (OEI1)</td>
<td>Unused</td>
</tr>
<tr>
<td>MTU6 and MTU7 output pin control setting</td>
<td></td>
</tr>
<tr>
<td>Enable MTIOC0B and MTIOC0D</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable MTIOC7A and MTIOC7C</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable MTIOC7B and MTIOC7D</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable POE4 input</td>
<td>Unused</td>
</tr>
<tr>
<td>Output enable interrupt 2 (OEI2)</td>
<td>Unused</td>
</tr>
<tr>
<td>GPT0, GPT1, GPT2 and GPT3 output pin control setting</td>
<td></td>
</tr>
<tr>
<td>Enable GTIOC0A and GTIOC0B</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable GTIOC1A and GTIOC1B</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable GTIOC2A and GTIOC2B</td>
<td>Unused</td>
</tr>
<tr>
<td>Code Generator Tool</td>
<td>3.API FUNCTIONS</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Enable GTIOC3A and GTIOC3B</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable POE10 input</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable POE11 input</td>
<td>Unused</td>
</tr>
<tr>
<td>Output enable interrupt 4 (OEI4)</td>
<td>Unused</td>
</tr>
<tr>
<td>Detection of stopped oscillation setting</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable request to place pins in the high-impedance on detection of stopped oscillation</td>
<td>Unused</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interrupt Controller Unit</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICU</td>
<td>Used</td>
</tr>
<tr>
<td>Group</td>
<td>Used</td>
</tr>
<tr>
<td>Group BL1</td>
<td>Used</td>
</tr>
<tr>
<td>Group BL1 Priority</td>
<td>Level 15 (highest)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Multi-Function Timer Pulse Unit 3</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTU3_U 0</td>
<td>Used</td>
</tr>
<tr>
<td>MTCLKA pin</td>
<td>Unused</td>
</tr>
<tr>
<td>MTCLKB pin</td>
<td>Unused</td>
</tr>
<tr>
<td>MTCLKC pin</td>
<td>Unused</td>
</tr>
<tr>
<td>MTCLKD pin</td>
<td>Unused</td>
</tr>
<tr>
<td>MTU0</td>
<td>Used</td>
</tr>
<tr>
<td>MTU0</td>
<td>Normal mode</td>
</tr>
<tr>
<td>Include this channel in the synchronous operation</td>
<td>Unused</td>
</tr>
<tr>
<td>Counter clock selection</td>
<td>PCLK</td>
</tr>
<tr>
<td>Counter clear source</td>
<td>Disabled counter clear</td>
</tr>
<tr>
<td>TGRA0 (Output compare register)</td>
<td>100μs (Actual value : 100)</td>
</tr>
<tr>
<td>TGRB0 (Output compare register)</td>
<td>100μs (Actual value : 100)</td>
</tr>
<tr>
<td>TGRC0 (Output compare register)</td>
<td>100μs (Actual value : 100)</td>
</tr>
<tr>
<td>TGRC0 (Output compare register)</td>
<td>100μs (Actual value : 100)</td>
</tr>
<tr>
<td>TGRE0 (Output compare register)</td>
<td>100μs (Actual value : 100)</td>
</tr>
<tr>
<td>TGRF0 (Output compare register)</td>
<td>100μs (Actual value : 100)</td>
</tr>
<tr>
<td>MTIOC0A pin (P34)</td>
<td>Initial output of MTIOC0A pin is 0. 0 output at compare match.</td>
</tr>
<tr>
<td>MTIOC0B pin (P13)</td>
<td>MTIOC0B pin output is disabled</td>
</tr>
<tr>
<td>MTIOC0C pin (P32)</td>
<td>Initial output of MTIOC0C pin is 0. 0 output at compare match.</td>
</tr>
<tr>
<td>MTIOC0D pin (P33)</td>
<td>MTIOC0D pin output is disabled</td>
</tr>
<tr>
<td>Enable A/D conversion start request on TGRA input capture/compare match (trigger signal of MTU0 TRGA0N)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable A/D conversion start request on TGRE compare match (trigger signal of MTU0 TRG0N)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable TGRA0 input capture/compare match interrupt (TGIA0)</td>
<td>Unused</td>
</tr>
<tr>
<td>----------------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>Enable TGRB0 input capture/compare match interrupt (TGIB0)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable TGRC0 input capture/compare match interrupt (TGIC0)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable TGRD0 input capture/compare match interrupt (TGID0)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable TGRE0 compare match interrupt (TGIE0)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable TGRF0 compare match interrupt (TGIF0)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable overflow interrupt (TCIVO)</td>
<td>Unused</td>
</tr>
</tbody>
</table>

**[API setting example]**

```c
r_cg_main.c

void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start the POE3 module */
    R_POE3_Start();
    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

```c
r_cg_poe3_user.c

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

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

Table 3.16 API Functions: [General PWM timer]

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

Performs initialization necessary to control the general PWM timer.

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_GPTn_Start**

Starts counting by a 16-bit timer.

**[Syntax]**

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

**Remark**  
\( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_GPTn_Stop**

Ends counting by a 16-bit timer.

**[Syntax]**

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

**Remark**

$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
### R_GPTn_HardwareStart

**Allows detection of GPT interrupts.**

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

**[Syntax]**

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

**Remark**  $n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_GPTn_HardwareStop

Prohibits detection of GPT interrupts.

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

Syntax

void R_GPTn_HardwareStop ( void );

Remark $n$ is the channel number.

Argument(s)

None.

Return value

None.
**R_GPT_Create_UserInit**

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

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_gpt_gtcmn_interrupt

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

[Syntax]
static void r_gpt_gtcmn_interrupt ( void );

Remark  
\( m \) is the timer general register number, \( n \) is the channel number.

[Argument(s)]
None.

[Return value]
None.
**r_gpt_gtcivn_interrupt**

Performs processing in response to the overflow interrupt.

**[Syntax]**

```c
static void r_gpt_gtcivn_interrupt ( void );
```

**Remark**  
\( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_gpt_gtcium_interrupt

Performs processing in response to the underflow interrupt.

**[Syntax]**

```
static void    r_gpt_gtcium_interrupt ( void );
```

Remark $n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_gpt_gdten_interrupt

Performs processing in response to the dead time error interrupt.

[Syntax]

static void r_gpt_gdten_interrupt ( void );

Remark \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
**r_gpt_etgip_interrupt**

Performs processing in response to the external trigger rising interrupt.

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_gpt_etgin_interrupt

Performs processing in response to the external trigger falling interrupt.

[Syntax]

static void r_gpt_etgin_interrupt ( void );

[Argument(s)]

None.

[Return value]

None.
Usage example

Use timer as One-shot timer.

[GUI setting example]

<table>
<thead>
<tr>
<th>General PWM Timer</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gp0</td>
<td>Used</td>
</tr>
<tr>
<td>GTETRG pin</td>
<td>Unused</td>
</tr>
<tr>
<td>GptChannel0</td>
<td>Used</td>
</tr>
<tr>
<td>GPT0</td>
<td>Saw-wave one-shot pulse mode</td>
</tr>
<tr>
<td>Clock source</td>
<td>PCLKA/8 0.75 (MHz)</td>
</tr>
<tr>
<td>Timer operation period</td>
<td>50 ms (Actual value: 50)</td>
</tr>
<tr>
<td>Period register value</td>
<td>37499</td>
</tr>
<tr>
<td>Buffer operation(GTPR)</td>
<td>Buffer operation is not performed</td>
</tr>
<tr>
<td>Count direction</td>
<td>Up-counting</td>
</tr>
<tr>
<td>Counter initial value</td>
<td>0</td>
</tr>
<tr>
<td>Counter clear</td>
<td>Disable</td>
</tr>
<tr>
<td>Hardware source for counter start</td>
<td>Disable</td>
</tr>
<tr>
<td>Hardware source for counter stop/clear</td>
<td>Disable</td>
</tr>
<tr>
<td>GTCCRA operation</td>
<td>Compare match</td>
</tr>
<tr>
<td>Compare match value (GTCCRA)</td>
<td>10</td>
</tr>
<tr>
<td>Buffer operation(GTCCRA)</td>
<td>Double buffer operation</td>
</tr>
<tr>
<td>GTIOC0A pin function</td>
<td>Disable</td>
</tr>
<tr>
<td>GTCCRB operation</td>
<td>Compare match</td>
</tr>
<tr>
<td>Compare match value (GTCCRB)</td>
<td>20</td>
</tr>
<tr>
<td>Buffer operation(GTCCRB)</td>
<td>Double buffer operation</td>
</tr>
<tr>
<td>GTIOC0B pin function</td>
<td>Disable</td>
</tr>
<tr>
<td>Automatically set GTCCRB0 using GTCCRA0 value and dead time</td>
<td>Unused</td>
</tr>
<tr>
<td>GTCCRC operation</td>
<td>Buffer register for GTCCRA</td>
</tr>
<tr>
<td>GTCCRD operation</td>
<td>Double buffer register for GTCCRA</td>
</tr>
<tr>
<td>GTCCRE operation</td>
<td>Buffer register for GTCCRB</td>
</tr>
<tr>
<td>GTCCRF operation</td>
<td>Double buffer register for GTCCRB</td>
</tr>
<tr>
<td>Enable compare match (up-counting) A/D conversion start request(GTADTRA)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable compare match (down-counting) A/D conversion start request(GTADTRA)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable compare match (up-counting) A/D conversion start request(GTADTRB)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable compare match (down-counting) A/D conversion start request(GTADTRB)</td>
<td>Unused</td>
</tr>
</tbody>
</table>
Enable GTCCRA input capture/compare match interrupt (GTCIA0)
Used Level 15 (highest)

Enable GTCCRB input capture/compare match interrupt (GTCIB0)
Used Level 15 (highest)

Enable GTCNT overflow (GTPR compare match) interrupt (GTCIV0)
Used Level 15 (highest)

[API setting example]

r_cg_main.c

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

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

r_cg_gpt_user.c

```c
static void r_gpt_gtciv0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop GPT channel 0 counter */
    R_GPT0_Stop();
    /* End user code. Do not edit comment generated here */
}
```
3.2.17 16-bit timer pulse unit (TPU)

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

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_TPU_Create</td>
<td>Performs initialization necessary to control the 16-bit timer pulse unit.</td>
</tr>
<tr>
<td>R_TPUStart</td>
<td>Starts counting by a 16-bit timer.</td>
</tr>
<tr>
<td>R_TPUPStop</td>
<td>Ends counting by a 16-bit timer.</td>
</tr>
<tr>
<td>R_TPU_Create_UserInit</td>
<td>Performs user-defined initialization relating to the 16-bit timer pulse unit.</td>
</tr>
<tr>
<td>r_tpu_tginm_interrupt</td>
<td>Performs processing in response to the input capture/compare match interrupt.</td>
</tr>
<tr>
<td>r_tpu_tcinv_interrupt</td>
<td>Performs processing in response to the overflow interrupt.</td>
</tr>
<tr>
<td>r_tpu_tcinu_interrupt</td>
<td>Performs processing in response to the underflow interrupt.</td>
</tr>
</tbody>
</table>
**R_TPU_Create**

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

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_TPU\_n\_Start**

Starts counting by a 16-bit timer.

**[Syntax]**

```c
void R_TPU\_n\_Start ( void );
```

Remark  \(n\) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_TPU\textsubscript{n} _Stop

Ends counting by a 16-bit timer.

**[Syntax]**

```c
void R_TPU\textsubscript{n} _Stop ( void );
```

Remark \textit{n} is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_TPU_Create_UserInit

Performs user-defined initialization relating to the 16-bit timer pulse unit.
Remark   This API function is called as the R_TPU_Create callback routine.

[Syntax]

void    R_TPU_Create_UserInit ( void );

[Argument(s)]
None.

[Return value]
None.
r_tpu_tginm_interrupt

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

[Syntax]

```
static void r_tpu_tginm_interrupt ( void );
```

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

[Argument(s)]

None.

[Return value]

None.
### r_tpu_tcinv_interrupt

Performs processing in response to the overflow interrupt.

**[Syntax]**

```
static void r_tpu_tcinv_interrupt ( void );
```

**Remark**

\( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_tpu_tcinu_interrupt

Performs processing in response to the underflow interrupt.

[Syntax]

```
static void r_tpu_tcinu_interrupt ( void );
```

Remark $n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
Usage example
Use timer as One-shot timer.

[GUI setting example]

<table>
<thead>
<tr>
<th>16-Bit Timer Pulse Unit</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPU_U0</td>
<td>Used</td>
</tr>
<tr>
<td>TCLKA pin</td>
<td>Unused</td>
</tr>
<tr>
<td>TCLKB pin</td>
<td>Unused</td>
</tr>
<tr>
<td>TCLKC pin</td>
<td>Unused</td>
</tr>
<tr>
<td>TCLKD pin</td>
<td>Unused</td>
</tr>
<tr>
<td>TPU0</td>
<td>Used</td>
</tr>
<tr>
<td>Include this channel in the synchronous operation</td>
<td>Unused</td>
</tr>
<tr>
<td>Counter clock selection</td>
<td>PCLK/64</td>
</tr>
<tr>
<td>Clock edge setting</td>
<td>Rising edge</td>
</tr>
<tr>
<td>Counter clear source</td>
<td>TGRA0 compare match/input capture (Use TGRA0 as a cycle register)</td>
</tr>
<tr>
<td>TGRA0 (Output compare register)</td>
<td>100ms (Actual value : 100)</td>
</tr>
<tr>
<td>TGRB0 (Output compare register)</td>
<td>100μs (Actual value : 96)</td>
</tr>
<tr>
<td>TGRD0 (Output compare register)</td>
<td>100μs (Actual value : 96)</td>
</tr>
<tr>
<td>TIOCA0 pin (PA0)</td>
<td>TIOCA0 pin output is disabled</td>
</tr>
<tr>
<td>TIOCB0 pin (PA1)</td>
<td>TIOCB0 pin output is disabled</td>
</tr>
<tr>
<td>TIOCC0 pin (P32)</td>
<td>TIOCC0 pin output is disabled</td>
</tr>
<tr>
<td>TIOCD0 pin (PA3)</td>
<td>TIOCD0 pin output is disabled</td>
</tr>
<tr>
<td>Enable A/D conversion start request on TGRA input capture/compare match (trigger signal of TPU0 TRGA0N)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable TGRA0 input capture/compare match interrupt (TG0A)</td>
<td>Level 15 (highest)</td>
</tr>
<tr>
<td>Enable TGRB0 input capture/compare match interrupt (TG0B)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable TGRD0 input capture/compare match interrupt (TG0D)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable overflow interrupt (TC10V)</td>
<td>Unused</td>
</tr>
</tbody>
</table>
[API setting example]

r_cg_main.c

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

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

r_cg_tpu_user.c

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

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_TMR_Create</td>
<td>Performs initialization necessary to control the 8-bit timer pulse unit.</td>
</tr>
<tr>
<td>R_TMRn_Start</td>
<td>Starts counting by an 8-bit timer.</td>
</tr>
<tr>
<td>R_TMRn_Stop</td>
<td>Ends counting by an 8-bit timer.</td>
</tr>
<tr>
<td>R_TMR_Create_UserInit</td>
<td>Performs user-defined initialization relating to the 8-bit timer pulse unit.</td>
</tr>
<tr>
<td>r_tmr_cmimn_interrupt</td>
<td>Performs processing in response to the compare match interrupt.</td>
</tr>
<tr>
<td>r_tmr_oviin_interrupt</td>
<td>Performs processing in response to the overflow interrupt.</td>
</tr>
</tbody>
</table>
R_TMR_Create

Performs initialization necessary to control the 8-bit timer.

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

[Syntax]

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

[Argument(s)]
None.

[Return value]
None.
R_TMRn_Start

Starts counting by an 8-bit timer.

[Syntax]

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

Remark  

\( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
**R_TMRn_Stop**

Ends counting by an 8-bit timer.

**[Syntax]**

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

Remark  $n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
### R_TMR_Create_UserInit

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

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_tmr_cmi\_m\_interrupt**

Performs processing in response to the compare match interrupt.

**[Syntax]**
```c
static void r_tmr_cmi\_m\_interrupt ( void );
```

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

**[Argument(s)]**
None.

**[Return value]**
None.
r_tmr_ovi_interrrupt

Performs processing in response to the overflow interrupt.

[Syntax]

static void r_tmr_ovi_interrupt ( void );

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.
Usage example

Use timer as One-shot timer.

[GUI setting example]

<table>
<thead>
<tr>
<th>8-Bit Timer</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmr0</td>
<td>Used</td>
</tr>
<tr>
<td>TmrChannel0</td>
<td>Used</td>
</tr>
<tr>
<td>TMR0</td>
<td>Used</td>
</tr>
<tr>
<td>Clock source</td>
<td>Used</td>
</tr>
<tr>
<td>Counter clear</td>
<td>Used</td>
</tr>
<tr>
<td>Compare match A value (TCORA)</td>
<td>Used</td>
</tr>
<tr>
<td>Compare match B value (TCORB)</td>
<td>Used</td>
</tr>
<tr>
<td>Enable TMO0 output</td>
<td>Used</td>
</tr>
<tr>
<td>Enable TCORA compare match interrupt (CMIA0)</td>
<td>Used</td>
</tr>
<tr>
<td>Enable TCORB compare match interrupt (CMIB0)</td>
<td>Used</td>
</tr>
<tr>
<td>Enable TCNT overflow interrupt (OV10)</td>
<td>Used</td>
</tr>
</tbody>
</table>

[API setting example]

r_cg_main.c

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

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

r_cg_tmr_user.c

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

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

Table 3.19 API Functions: [Programmable Pulse Generator]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_PPG_Create</td>
<td>Performs initialization necessary to control the programmable pulse generator.</td>
</tr>
<tr>
<td>R_PPG_Create_UserInit</td>
<td>Performs user-defined initialization relating to the programmable pulse generator.</td>
</tr>
</tbody>
</table>
R_PPG_Create

Performs initialization necessary to control the programmable pulse generator.

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

[Syntax]

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

[Argument(s)]
None.

[Return value]
None.
**R_PPG_Create_UserInit**

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

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
### 3.2.20 Compare match timer (CMT)

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>R_CMTn_Create</code></td>
<td>Performs initialization necessary to control the compare match timer.</td>
</tr>
<tr>
<td><code>R_CMTn_Create_UserInit</code></td>
<td>Performs user-defined initialization relating to the compare match timer.</td>
</tr>
<tr>
<td><code>r_cmt_cmin_interrupt</code></td>
<td>Performs processing in response to the compare match interrupt (CMI[n]).</td>
</tr>
<tr>
<td><code>R_CMTn_Start</code></td>
<td>Starts counting by a 16-bit timer.</td>
</tr>
<tr>
<td><code>R_CMTn_Stop</code></td>
<td>Ends counting by a 16-bit timer.</td>
</tr>
</tbody>
</table>
R_CMTn_Create

Performs initialization necessary to control the compare match timer.

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

[Syntax]

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

Remark \( n \) is the channel number.

[Argument(s)]
None.

[Return value]
None.
**R_CMTn_Create_UserInit**

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

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

**[Syntax]**

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

**Remark**  
$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_cmt_cmi_interrupt**

Performs processing in response to the compare match interrupt (CMI\(_n\)).

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

**[Syntax]**

```c
static void  r_cmt_cmi_interrupt ( void );
```

**Remark**  
\( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_CMTn_Start

Starts counting by a 16-bit timer.

[Syntax]

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

Remark  

$n$ is the channel number.

[Argument(s)]
None.

[Return value]
None.
**R_CMTn_Stop**

Ends counting by a 16-bit timer.

**[Syntax]**

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

**Remark**  
$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
Usage example

Use timer as One-shot timer.

[GUI setting example]

<table>
<thead>
<tr>
<th></th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compare Match Timer</strong></td>
<td></td>
</tr>
<tr>
<td><strong>CMT0</strong></td>
<td></td>
</tr>
<tr>
<td>Compare match timer operation setting</td>
<td>Used</td>
</tr>
<tr>
<td>Count clock setting</td>
<td>PCLK/512</td>
</tr>
<tr>
<td>Interval value setting</td>
<td>100ms (Actual value : 100.010667)</td>
</tr>
<tr>
<td>Enable compare match interrupt (CMI0)</td>
<td>Used</td>
</tr>
<tr>
<td>Priority</td>
<td>Level 15 (highest)</td>
</tr>
</tbody>
</table>

[API setting example]

```c
r_cg_main.c

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

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

r_cg_cmt_user.c

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

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_CMTWn_Create</td>
<td>Performs initialization necessary to control the compare match timer W.</td>
</tr>
<tr>
<td>R_CMTWn_Start</td>
<td>Starts counting by a 16-bit timer.</td>
</tr>
<tr>
<td>R_CMTWn_Stop</td>
<td>Ends counting by a 16-bit timer.</td>
</tr>
<tr>
<td>R_CMTWn&gt;Create_UserInit</td>
<td>Performs user-defined initialization relating to the compare match timer W.</td>
</tr>
<tr>
<td>r_cmtw_cmwin_interrupt</td>
<td>Performs processing in response to the compare match interrupt.</td>
</tr>
<tr>
<td>r_cmtw_icmin_interrupt</td>
<td>Performs processing in response to the input capture interrupt.</td>
</tr>
<tr>
<td>r_cmtw_ocmin_interrupt</td>
<td>Performs processing in response to the output compare interrupt.</td>
</tr>
</tbody>
</table>
**R_CMTWn_Create**

Performs initialization necessary to control the compare match timer W.

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

**[Syntax]**

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

Remark $n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_CMTWn_Start**

Starts counting by a 16-bit timer.

**[Syntax]**

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

Remark  
$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_CMTWn_Stop**

Ends counting by a 16-bit timer.

### [Syntax]

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

*Remark*  

$n$ is the channel number.

### [Argument(s)]

None.

### [Return value]

None.
R_CMTWn_Create_UserInit

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

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

[Syntax]

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

Remark $n$ is the channel number.

[Argument(s)]
None.

[Return value]
None.
r_cmtw_cmwin_interrupt

Performs processing in response to the compare match interrupt.

[Syntax]

static void r_cmtw_cmwin_interrupt ( void );

Remark  \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
r_cmtw_icmin_interrupt

Performs processing in response to the input capture interrupt.

[Syntax]

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

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

[Argument(s)]
None.

[Return value]
None.
r_cmtw_ocmin_interrupt

Performs processing in response to the output compare interrupt.

[Syntax]

```c
static void r_cmtw_ocmin_interrupt ( void );
```

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

[Argument(s)]
None.

[Return value]
None.
Usage example

Use timer as One-shot timer.

[GUI setting example]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compare match timer W</td>
<td>Used</td>
</tr>
<tr>
<td>CMTW0</td>
<td></td>
</tr>
<tr>
<td>Compare match timer W operation</td>
<td>Used</td>
</tr>
<tr>
<td>setting</td>
<td></td>
</tr>
<tr>
<td>Count Clock setting</td>
<td>PCLK/8</td>
</tr>
<tr>
<td>Timer Counter size</td>
<td>32 bits</td>
</tr>
<tr>
<td>Compare match setting</td>
<td></td>
</tr>
<tr>
<td>interval value</td>
<td>100ms (Actual value : 100)</td>
</tr>
<tr>
<td>Register value (CMWCCOR)</td>
<td>74999</td>
</tr>
<tr>
<td>Compare match interrupt (CMWII0)</td>
<td>Used</td>
</tr>
<tr>
<td>Priority (CMWII0)</td>
<td>Level 15 (highest)</td>
</tr>
<tr>
<td>Counter clear</td>
<td>CMWCNT is cleared by CMWCCOR</td>
</tr>
<tr>
<td>TIC0 pin</td>
<td>Used</td>
</tr>
<tr>
<td>Input capture control 0</td>
<td>Rising edge</td>
</tr>
<tr>
<td>Enable input capture interrupt (IC0I0)</td>
<td>Used</td>
</tr>
<tr>
<td>Priority (IC0I0)</td>
<td>Level 15 (highest)</td>
</tr>
<tr>
<td>TIC1 pin</td>
<td>Used</td>
</tr>
<tr>
<td>Input capture control 1</td>
<td>Rising edge</td>
</tr>
<tr>
<td>Enable input capture interrupt (IC1I0)</td>
<td>Used</td>
</tr>
<tr>
<td>Priority (IC1I0)</td>
<td>Level 15 (highest)</td>
</tr>
<tr>
<td>TOC0 pin</td>
<td>Used</td>
</tr>
<tr>
<td>Output compare control 0</td>
<td>Retains the output value</td>
</tr>
<tr>
<td>Compare value 0</td>
<td>10</td>
</tr>
<tr>
<td>Enable output compare interrupt (OC0I0)</td>
<td>Used</td>
</tr>
<tr>
<td>Priority (OC0I0)</td>
<td>Level 15 (highest)</td>
</tr>
<tr>
<td>TOC1 pin</td>
<td>Used</td>
</tr>
<tr>
<td>Output compare control 1</td>
<td>Retains the output value</td>
</tr>
<tr>
<td>Compare value 1</td>
<td>20</td>
</tr>
<tr>
<td>Enable output compare interrupt (OC1I0)</td>
<td>Used</td>
</tr>
<tr>
<td>Priority (OC1I0)</td>
<td>Level 15 (highest)</td>
</tr>
</tbody>
</table>
[API setting example]

r_cg_main.c

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

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

r_cg_cmtw_user.c

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

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

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

Performs initialization necessary to control the realtime clock.

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_RTC_Create_UserInit**

Performs user-defined initialization relating to the realtime clock.

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_rtc alm interrupt**

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

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

**[Syntax]**

```c
static void r_rtc alm interrupt ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
r_rtc_prd_interrupt

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

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
r_rtc_cup_interrupt

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

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

[Syntax]

static void r_rtc_cup_interrupt ( void );

[Argument(s)]

None.

[Return value]

None.
R_RTC_Set_CalendarAlarm

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

[Syntax]

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

[Argument(s)]

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

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

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

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

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

[Return value]

None.
R_RTC_Set_BinaryAlarm

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

[Syntax]

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

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint32_t alarm_enable;</td>
<td>Comparison flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x0 : Comparison does not proceed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x1 : Comparison proceeds</td>
</tr>
<tr>
<td>I</td>
<td>uint32_t alarm_val;</td>
<td>The value of the binary counter</td>
</tr>
</tbody>
</table>

[Return value]

None.
R_RTC_Set_ConstPeriodInterruptOn

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

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

[Argument(s)]

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

[Return value]
None.
R_RTC_Set_ConstPeriodInterruptOff

Prohibits detection of the periodic interrupt (PRD).

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
R_RTC_Set_CarryInterruptOn

Allows detection of the carry interrupt (CUP).

[Syntax]

void R_RTC_Set_CarryInterruptOn ( void );

[Argument(s)]
None.

[Return value]
None.
R_RTC_Set_CarryInterruptOff

Prohibits detection of the carry interrupt (CUP).

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
R_RTC_Set_RTCOUTOn

Sets the RTCOUT output period and starts RTCOUT output.

[Syntax]

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

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>rtc_rtcout_period_t rtcout_freq;</td>
<td>RTCOUT output period</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RTCOUT_1HZ : 1Hz</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RTCOUT_64HZ : 64Hz</td>
</tr>
</tbody>
</table>

[Return value]

None.
R_RTC_Set_RTCOUTOff

Ends RTCOUT output.

[Syntax]

void    R_RTC_Set_RTCOUTOff ( void );

[Argument(s)]

None.

[Return value]

None.
R_RTC_Start

Starts counting.

[Syntax]

void R_RTC_Start ( void );

[Argument(s)]

None.

[Return value]

None.
R_RTC_Stop

Ends counting.

[Syntax]

    void R_RTC_Stop ( void );

[Argument(s)]

    None.

[Return value]

    None.
R_RTC_Restart

Initializes the counter then starts counting.

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

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

[Syntax]

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

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>rtc_calendarcounter_value_t counter_write_val;</td>
<td>Initial value (year, month, date, day-of-week, time, minute, and second)</td>
</tr>
</tbody>
</table>

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

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

[Return value]

None.
Sets the calendar and time values.

**Syntax**

```c
#include "r_cg_rtc.h"

void R_RTC_Set_CalendarCounterValue ( rtc_calendarcounter_value_t counter_write_val );
```

**Argument(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>rtc_calendarcounter_value_t</td>
<td>Calendar and time values (year, month, date, day-of-week, time, minute, and second)</td>
</tr>
</tbody>
</table>

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

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

**Return value**

None.
R_RTC_Get_CalendarCounterValue

Gets the calendar and time values.

[Syntax]

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

[Argument(s)]

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

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

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

[Return value]
None.
**R_RTC_Set_BinaryCounterValue**

Sets the value of the binary counter.

**[Syntax]**

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

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint32_t counter_write_val;</td>
<td>The value of the binary counter</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
**R_RTC_Get_BinaryCounterValue**

Gets the value of the binary count.

**[Syntax]**

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

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint32_t * const counter_read_val</td>
<td>Pointer to an area where the obtained value of the binary counter is to be stored</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
**R_RTC_Get_CalendarTimeCaptureValue**

Gets the captured calendar time value.

**[Syntax]**

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

Remark  
*n* is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>rtc_calendarcounter_value_t * const</td>
<td>Pointer to the area where the obtained calendar and time values.</td>
</tr>
</tbody>
</table>

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

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

**[Return value]**

None.
R_RTC_Get_BinaryTimeCaptureValue

Gets the value of the binary count.

[Syntax]

```c
void R_RTC_Get_BinaryTimeCaptureValue ( uint32_t * const counter_read_val );
```

Remark

$n$ is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint32_t * const counter_read_val</td>
<td>The value of the binary counter.</td>
</tr>
</tbody>
</table>

[Return value]

None.
Usage example

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

[GUI setting example]

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

r_cg_main.c

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

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

r_cg_rtc_user.c

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

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

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

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

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

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

Table 3.23 API Functions: [Watchdog Timer]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_WDT_Create</td>
<td>Performs initialization necessary to control the watchdog timer.</td>
</tr>
<tr>
<td>R_WDT_Restart</td>
<td>Clears the watchdog timer counter and resumes counting.</td>
</tr>
<tr>
<td>R_WDT_Create_UserInit</td>
<td>Performs user-defined initialization relating to the watchdog timer.</td>
</tr>
<tr>
<td>r_wdt_wuni_interrupt</td>
<td>Performs processing in response to the non-maskable/maskable interrupt.</td>
</tr>
<tr>
<td>r_wdt_nmi_interrupt</td>
<td>Performs processing in response to the non-maskable interrupt.</td>
</tr>
</tbody>
</table>
**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]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_WDT_Restart

Clears the watchdog timer counter and resumes counting.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
**R_WDT_Create_UserInit**

Performs user-defined initialization relating to the watchdog timer.

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_wdt_wuni_interrupt

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

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

[Syntax]

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

[Argument(s)]
None.

[Return value]
None.
r_wdt_nmi_interrupt

Performs processing in response to the non-maskable interrupt.

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

[Syntax]

static void r_wdt_nmi_interrupt ( void );

[Argument(s)]
None.

[Return value]
None.
Usage example

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

[GUI setting example]

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

[API setting example]

r_cg_main.c

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

r_cg_wdt_user.c

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

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

Table 3.24 API Functions: [Independent Watchdog Timer]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_IWDT_Create</td>
<td>Performs initialization necessary to control the independent watchdog timer.</td>
</tr>
<tr>
<td>R_IWDT_Create_UserInit</td>
<td>Performs user-defined initialization relating to the independent watchdog timer.</td>
</tr>
<tr>
<td>r_iwdt_nmi_interrupt</td>
<td>Performs processing in response to the non-maskable interrupt (WUNI).</td>
</tr>
<tr>
<td>r_iwdt_iwuni_interrupt</td>
<td>Performs processing in response to the non-maskable/maskable interrupt.</td>
</tr>
<tr>
<td>R_IWDT_Restart</td>
<td>clears the independent watchdog timer counter and resumes counting.</td>
</tr>
</tbody>
</table>
**R_IWDT_Create**

Performs initialization necessary to control the independent watchdog timer.

**Remark**
This API function is called from `R_SystemInit` before `main()` is executed.

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_IWDT_Create_UserInit

Performs user-defined initialization relating to the independent watchdog timer.
Remark This API function is called as the R_IWDT_Create callback routine.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
**r_iwdt_nmi_interrupt**

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

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_iwdt_iwuni_interrupt**

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

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_IWDT_Restart

Clears the independent watchdog timer counter and resumes counting.

[Syntax]

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

[Argument(s)]
None.

[Return value]
None.
Usage example

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

[GUI setting example]

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

[API setting example]

r_cg_main.c

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

r_cg_iwdt_user.c

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

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

**Table 3.25 API Functions: [Serial Communications Interface]**

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

Performs initialization necessary to control the serial communication interface.
Remark This API function is called from R_Systeminit before main() is executed.

[Syntax]

```c
void R_SCI{n}_Create ( void );
```

Remark \(n\) is the channel number.

[Argument(s)]
None.

[Return value]
None.
**R_SCI\textsubscript{n}_Create_UserInit**

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

Remark This API function is called as the \texttt{R_SCI\textsubscript{n}_Create} callback routine.

**[Syntax]**

\[
\text{void } \texttt{R_SCI}_n\_Create\_UserInit ( \text{void} );
\]

Remark \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_sci_transmitend_interrupt**

Performs processing in response to the transmit-end interrupts.

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

**[Syntax]**

```c
static void r_sci_transmitend_interrupt ( void );
```

**Remark** \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_sci_transmitinterrupt**

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

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

**[Syntax]**

```c
static void r_sci_transmitinterrupt ( void );
```

**Remark** \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_sci_receive_interrupt**

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

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

**[Syntax]**

```c
static void r_sci_receive_interrupt ( void );
```

*Remark* $n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_sci_receiveerror_interrupt

Performs processing in response to the receive error interrupts.

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

[Syntax]

    static void r_sci_receiveerror_interrupt ( void );

Remark   \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_SCI\textunderscore n\_Start

Starts SCI communication.

[Syntax]

\begin{verbatim}
void R_SCI\textunderscore n\_Start ( void );
\end{verbatim}

Remark \( n \) is the channel number.

[Argument(s)]
None.

[Return value]
None.
R_SCI\textsubscript{n} Stop

Ends SCI communication.

[Syntax]

\begin{verbatim}
void R_SCI\textsubscript{n} Stop ( void );
\end{verbatim}

Remark \textit{n} is the channel number.

[Argument(s)]

None.

[Return value]

None.
**R_SCln_Serial_Send**

Starts SCI transmission (asynchronous mode).

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

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

**[Syntax]**

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

Remark  * n is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint8_t * const tx_buf;</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t tx_num;</td>
<td>Total amount of data to send</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument <code>tx_num</code> specification</td>
</tr>
</tbody>
</table>
R_SCI{n}_Serial_Receive

Starts SCI reception (asynchronous mode).

Remark 1. This API function repeats SCI reception in byte units the number of times specified by the argument \textit{rx\_num} and then stores the received data in the buffer at the location specified by the argument \textit{rx\_buf}.

Remark 2. When performing a SCI reception, \texttt{R\_SCIn\_Start} must be called before this API function is called.

**[Syntax]**

```c
#include "r_cg_macrodriver.h"

MD_STATUS R_SCI{n}_Serial_Receive ( uint8_t * const rx_buf, uint16_t rx_num );
```

Remark \( n \) is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint8_t * const rx_buf;</td>
<td>Pointer to a buffer to store the reception data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t rx_num;</td>
<td>Total amount of data to receive</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument \textit{rx_num} specification</td>
</tr>
</tbody>
</table>

R_SCIn_Serial_Multiprocessor_Send

Starts SCI transmission (multi-processor communications function).

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

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

[Syntax]

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

Remark  n is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint8_t * const id_buf;</td>
<td>Pointer to a buffer storing the transmission ID</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t id_num;</td>
<td>Total amount of ID to send</td>
</tr>
<tr>
<td>I</td>
<td>uint8_t * const tx_buf;</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t tx_num;</td>
<td>Total amount of data to send</td>
</tr>
</tbody>
</table>

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument tx_num specification</td>
</tr>
</tbody>
</table>
### R_SCI\_n\_Serial\_Multiprocessor\_Receive

Starts SCI reception (multi-processor communications function).

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

**Remark 2.** When performing a SCI reception, `R_SCI\_n\_Start` must be called before this API function is called.

#### [Syntax]

```c
#include "r_cg_macrodriver.h"

MD_STATUS R_SCI\_n\_Serial\_Multiprocessor\_Receive ( uint8_t * const rx_buf, uint16_t rx_num );
```

**Remark**  \( n \) is the channel number.

#### [Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint8_t * const rx_buf;</td>
<td>Pointer to a buffer to store the reception data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t rx_num;</td>
<td>Total amount of data to receive</td>
</tr>
</tbody>
</table>

#### [Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument <code>rx_num</code> specification</td>
</tr>
</tbody>
</table>
**R_SCI\textsubscript{n} Serial_Send_Receive**

Starts SCI transmission/reception (clock synchronous mode).

**Remark 1.** This API function repeats SCI transmission in byte units the number of times specified by the argument \textit{tx\_num} from the buffer at the location specified by the argument \textit{tx\_buf}.

**Remark 2.** This API function repeats SCI reception processing in byte units the number of times specified by the argument \textit{rx\_num} and then stores the received data in the buffer at the location specified by the argument \textit{rx\_buf}.

**Remark 3.** When performing a SCI transmission/reception, \texttt{R\_SCI\textsubscript{n} Start} must be called before this API function is called.

**[Syntax]**

```c
#include "r_cg_macrodriver.h"
MD_STATUS R_SCI\textsubscript{n} Serial_Send_Receive ( uint8_t * const tx\_buf, uint16_t tx\_num, uint8_t * const rx\_buf, uint16_t rx\_num );
```

Remark \( n \) is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint8_t * const tx_buf;</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t tx_num;</td>
<td>Total amount of data to send</td>
</tr>
<tr>
<td>O</td>
<td>uint8_t * const rx_buf;</td>
<td>Pointer to a buffer to store the reception data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t rx_num;</td>
<td>Total amount of data to receive</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument \textit{tx_num} specification</td>
</tr>
</tbody>
</table>
**R_SCI\textsubscript{n} SmartCard\_Send**

Starts SCI transmission (smart card interface mode).

Remark 1. This API function repeats the byte-level SCI transmission from the buffer specified in argument \textit{tx\_buf} the number of times specified in argument \textit{tx\_num}.

Remark 2. When performing a SCI transmission, \textit{R\_SCI\textsubscript{n} Start} must be called before this API function is called.

**[Syntax]**

```c
#include "r_cg_macrodriver.h"
MD\_STATUS \textit{R\_SCI\textsubscript{n} SmartCard\_Send} ( uint8\_t * const \textit{tx\_buf}, uint16\_t \textit{tx\_num} );
```

Remark \( n \) is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint8_t * const \textit{tx_buf};</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t \textit{tx_num};</td>
<td>Total amount of data to send</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument \textit{tx_num} specification</td>
</tr>
</tbody>
</table>
**R_SCln_SmartCard_Receive**

Starts SCI reception (smart card interface mode).

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

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

**[Syntax]**

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

Remark  `n` is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint8_t * const rx_buf;</td>
<td>Pointer to a buffer to store the reception data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t rx_num;</td>
<td>Total amount of data to receive</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument <code>rx_num</code> specification</td>
</tr>
</tbody>
</table>
**R_SCI\textsubscript{n} IIC\_Master\_Send**

Starts SCI master transmission (simple I2C mode).

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

**Remark 2.** This API function internally calls \texttt{R\_SCIn\_IIC\_StartCondition} to handle processing to start SCI master transmission.

**Remark 3.** When performing a SCI master transmission, \texttt{R\_SCIn\_Start} must be called before this API function is called.

**[Syntax]**

```c
void R_SCI\textsubscript{n} IIC\_Master\_Send ( uint8_t \textit{adr}, uint8_t * const \textit{tx\_buf}, uint16_t \textit{tx\_num} );
```

Remark \textit{n} is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint8_t \textit{adr};</td>
<td>Slave address</td>
</tr>
<tr>
<td>I</td>
<td>uint8_t * const \textit{tx_buf};</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t \textit{tx_num};</td>
<td>Total amount of data to send</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
**R_SCI\_IIC\_Master\_Receive**

Starts SCI master reception (simple I2C mode).

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

**Remark 2.** This API function internally calls R\_SCIn\_IIC\_StartCondition to handle processing to start SCI master reception.

**Remark 3.** When performing a SCI master reception, R\_SCIn\_Start must be called before this API function is called.

**[Syntax]**

```c
void    R_SCI\_IIC\_Master\_Receive ( uint8_t adr, uint8_t * const rx_buf, uint16_t rx_num );
```

Remark  
*n* is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint8_t adr;</td>
<td>Slave address</td>
</tr>
<tr>
<td>O</td>
<td>uint8_t * const rx_buf;</td>
<td>Pointer to a buffer to store the reception data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t rx_num;</td>
<td>Total amount of data to receive</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
**R_SCI\_n\_SPI\_Master\_Send**

Starts SCI master transmission (simple SPI mode).

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

**Remark 2.** When performing a SCI master transmission, `R_SCI\_n\_Start` must be called before this API function is called.

**[Syntax]**

```c
#include "r_cg_macrodriver.h"
MD_STATUS R_SCI\_n\_SPI\_Master\_Send ( uint8_t * const tx_buf, uint16_t tx_num );
```

Remark \(n\) is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint8_t * const tx_buf;</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t tx_num;</td>
<td>Total amount of data to send</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument (tx_num) specification</td>
</tr>
</tbody>
</table>
**R_SCI\_n\_SPI\_Master\_Send\_Receive**

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

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

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

**Remark 3.** When performing a SCI master transmission/reception, `R_SCI\_Start` must be called before this API function is called.

**[Syntax]**

```
#include       "r_cg_macrodriver.h"
MD_STATUS   R_SCI\_n\_SPI\_Master\_Send\_Receive ( uint8_t * const tx_buf, uint16_t tx_num,
                                                uint8_t * const rx_buf, uint16_t rx_num);
```

Remark  $n$ is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint8_t * const tx_buf;</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t tx_num;</td>
<td>Total amount of data to send</td>
</tr>
<tr>
<td>O</td>
<td>uint8_t * const rx_buf;</td>
<td>Pointer to a buffer to store the reception data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t rx_num;</td>
<td>Total amount of data to receive</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument <code>tx_num</code> specification</td>
</tr>
</tbody>
</table>
R_SCI\textsubscript{n} SPI_Slave_Send

Starts SCI slave transmission (simple SPI mode).

Remark 1. This API function repeats the byte-level SCI slave transmission from the buffer specified in argument \textit{tx\_buf} the number of times specified in argument \textit{tx\_num}.

Remark 2. When performing a SCI slave transmission, \texttt{R_SCI\textsubscript{n} Start} must be called before this API function is called.

[Syntax]

```c
#include "r_cg_macrodriver.h"
MD_STATUS R_SCI\textsubscript{n} SPI_Slave_Send (uint8_t * const tx\_buf, uint16_t tx\_num);
```

Remark \textit{n} is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint8_t * const tx_buf;</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t tx_num;</td>
<td>Total amount of data to send</td>
</tr>
</tbody>
</table>

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument \textit{tx_num} specification</td>
</tr>
</tbody>
</table>

R_SCI\textsubscript{n} SPI Slave Send Receive

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

Remark 1. This API function repeats SCI slave transmission in byte units the number of times specified by the argument \textit{tx\_num} from the buffer at the location specified by the argument \textit{tx\_buf}.

Remark 2. This API function repeats SCI slave reception in byte units the number of times specified by the argument \textit{rx\_num} and the received data are stored in the buffer at the location specified by the argument \textit{rx\_buf}.

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

[Syntax]

```c
#include "r_cg_macrodriver.h"

MD\_STATUS R_SCI\textsubscript{n} SPI\_Slave\_Send\_Receive ( uint8\_t * const tx\_buf, uint16\_t tx\_num,
uint8\_t * const rx\_buf, uint16\_t rx\_num );
```

Remark \textit{n} is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint8_t * const tx_buf; Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t tx_num; Total amount of data to send</td>
</tr>
<tr>
<td>O</td>
<td>uint8_t * const rx_buf; Pointer to a buffer to store the reception data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t rx_num; Total amount of data to receive</td>
</tr>
</tbody>
</table>

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument tx_num specification</td>
</tr>
</tbody>
</table>
R_SCI\textsubscript{n} IIC\_StartCondition

Sends the start bit.

Remark This API function is called as the internal function of \texttt{R\_SCIn\_IIC\_Master\_Send} and \texttt{R\_SCIn\_IIC\_Master\_Receive}.

### Syntax

```c
void R_SCI\textsubscript{n} IIC\_StartCondition ( void );
```

Remark \( n \) is the channel number.

### Argument(s)

None.

### Return value

None.
R_SCI\textsubscript{n}IIC_StopCondition

Sends the stop bit.

[Syntax]
\begin{verbatim}
void    R_SCI\textsubscript{n}IIC_StopCondition ( void );
\end{verbatim}

Remark \( n \) is the channel number.

[Argument(s)]
None.

[Return value]
None.
r_sci_callback_transmitend

Performs processing in response to the transmit-end interrupts.
Remark This API function is called as the r_scin_transmitend_interrupt callback routine.

[Syntax]
static void r_sci_callback_transmitend ( void );
Remark n is the channel number.

[Argument(s)]
None.

[Return value]
None.
r_sci_callback_receiveend

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

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

[Syntax]

```
static void r_sci_callback_receiveend ( void );
```

Remark $n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
r_sci_callback_receiveerror

Performs processing in response to the receive error interrupts.

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

[Syntax]

static void r_sci_callback_receiveerror ( void );

Remark $n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
### Usage example

Transmit capital letter 'A' by UART.

#### [GUI setting example]

<table>
<thead>
<tr>
<th>Serial Interface</th>
<th>Communications</th>
<th>Function setting</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCI6</td>
<td></td>
<td>Asynchronous mode (Transmission)</td>
<td></td>
</tr>
<tr>
<td>TXD6</td>
<td></td>
<td>P00</td>
<td></td>
</tr>
<tr>
<td>AsynchronousMode_Transmit6</td>
<td>Used</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data length setting</th>
<th>8 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parity setting</td>
<td>None</td>
</tr>
<tr>
<td>Stop bit length setting</td>
<td>1 bit</td>
</tr>
<tr>
<td>Transfer direction setting</td>
<td>LSB-first</td>
</tr>
<tr>
<td>Transfer clock</td>
<td>Internal clock</td>
</tr>
<tr>
<td>Bit rate</td>
<td>9600 (bps)</td>
</tr>
<tr>
<td>Enable modulation duty correction</td>
<td>Unused</td>
</tr>
<tr>
<td>SCK6 pin function</td>
<td>SCK6 is not used</td>
</tr>
<tr>
<td>Hardware flow control setting</td>
<td>None</td>
</tr>
<tr>
<td>Transmit data handling</td>
<td>Data handled in interrupt service routine</td>
</tr>
<tr>
<td>TX6 Priority</td>
<td>Level 15 (highest)</td>
</tr>
<tr>
<td>TE6 Priority (Group BL0)</td>
<td>(Please set priority setting in ICU)</td>
</tr>
<tr>
<td>Transmission end</td>
<td>Used</td>
</tr>
</tbody>
</table>

#### Interrupt Controller Unit

<table>
<thead>
<tr>
<th>Interrupt Controller Unit</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICU</td>
<td>Used</td>
</tr>
<tr>
<td>Group</td>
<td>Used</td>
</tr>
<tr>
<td>Group BL0</td>
<td>Used</td>
</tr>
<tr>
<td>Group BL0 priority</td>
<td>Level 15 (highest)</td>
</tr>
</tbody>
</table>
[API setting example]

r_cg_main.c

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

void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start the SCI6 channel */
    R_SCI6_Start();
    /* Transmit SCI6 data */
    R_SCI6_Serial_Send((uint8_t *)&g_sci6_tx_buf, 1U);
    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}

r_cg_sci_user.c

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

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

static void r_sci6_callback_transmitend(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop the SCI6 channel */
    R_SCI6_Stop();
    /* End user code. Do not edit comment generated here */
}
### 3.2.26 FIFO embedded serial communications interface (SCIFA)

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

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

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

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

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

**[Syntax]**

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

**Remark**  `n` is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_SCIFAn_Start

Starts FIFO embedded SCI communication.

[Syntax]

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

Remark  
$n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
**R_SCIFAn_Stop**

Ends FIFO embedded SCI communication.

**[Syntax]**

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

Remark  

$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
### R_SCIFAn_Serial_Send

Starts FIFO embedded SCI transmission (asynchronous mode).

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

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

**[Syntax]**

```c
#include "r_cg_macrodriver.h"

MD_STATUS R_SCIFAn_Serial_Send ( uint8_t * const tx_buf, uint16_t tx_num);
```

Remark  
`n` is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint8_t * const tx_buf;</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t tx_num;</td>
<td>Total amount of data to send</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument</td>
</tr>
</tbody>
</table>
R_SCIFAn_Serial_Receive

Starts FIFO embedded SCI reception (asynchronous mode).

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

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

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

Remark n is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint8_t * const rx_buf;</td>
<td>Pointer to a buffer to store the reception data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t rx_num;</td>
<td>Total amount of data to receive</td>
</tr>
</tbody>
</table>

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument</td>
</tr>
</tbody>
</table>
### R_SCIFA_n_Serial_Send_Receive

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

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

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

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

#### [Syntax]

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

Remark: `n` is the channel number.

#### [Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint8_t * const tx_buf;</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t tx_num;</td>
<td>Total amount of data to send</td>
</tr>
<tr>
<td>O</td>
<td>uint8_t * const rx_buf;</td>
<td>Pointer to a buffer to store the reception data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t rx_num;</td>
<td>Total amount of data to receive</td>
</tr>
</tbody>
</table>

#### [Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument</td>
</tr>
</tbody>
</table>
**R_SCIFAn_Create_UserInit**

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

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_scifan_teif_interrupt

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

[Syntax]

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

Remark $n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
r_scifan_txif_interrupt

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

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

[Syntax]
static void r_scifan_txif_interrupt ( void );

Remark  \( n \) is the channel number.

[Argument(s)]
None.

[Return value]
None.
**r_scifan_rxif_interrupt**

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

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

**[Syntax]**

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

Remark $n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_scifan_erif_interrupt

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

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

[Syntax]

static void r_scifan_erif_interrupt ( void );

Remark   \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
Performs processing in response to the break or overrun interrupts.

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

**[Syntax]**

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

Remark $n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
### r_scifan_drif_interrupt

Performs processing in response to the receive data ready interrupts.

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

**[Syntax]**

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

**Remark** 

$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_scifan_callback_transmitend

Performs processing in response to the transmit-end interrupts.

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

[Syntax]

static void r_scifan_callback_transmitend ( void );

Remark  \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
### r_scifan_callback_receiveend

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

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

**[Syntax]**

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

**Remark** `n` is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_scifan_callback_error

Performs processing in response to the error interrupts.

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

[Syntax]

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

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.
### Usage example

Transmit capital letter 'A' by UART.

#### [GUI setting example]

<table>
<thead>
<tr>
<th>Serial Communications</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface with FIFO</td>
<td>Used</td>
</tr>
<tr>
<td>SCIF9</td>
<td>Used</td>
</tr>
<tr>
<td>Function setting</td>
<td>Asynchronous mode (Transmission)</td>
</tr>
<tr>
<td>TXD9</td>
<td>PB7</td>
</tr>
<tr>
<td>AsynchronousMode_Transmit9</td>
<td>Used</td>
</tr>
</tbody>
</table>

- **Data length setting**: 8 bits
- **Parity setting**: None
- **Stop bit length setting**: 1 bit
- **Transfer direction setting**: LSB-first
- **Bit rate**: 9600 (bps)
- **Enable modulation duty correction**: Unused
- **SCK9 pin function**: SCK9 is not used
- **Enable modem control**: Unused
- **Transmit FIFO data trigger number**: 0
- **Transmit data handling**: Data handled in interrupt service routine
- **TX19 priority**: Level 15 (highest)
- **TE19 priority (Group AL0)**: (Please set priority setting in ICU)
- **Transmission end**: Used1

<table>
<thead>
<tr>
<th>Interrupt Controller Unit</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICU</td>
<td>Used</td>
</tr>
<tr>
<td>Group</td>
<td>Used</td>
</tr>
<tr>
<td>Group AL0</td>
<td>Used</td>
</tr>
<tr>
<td>Group AL0 priority</td>
<td>Level 15 (highest)</td>
</tr>
</tbody>
</table>
[API setting example]

r_cg_main.c

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

void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start the SCIFA9 channel */
    R_SCIFA9_Start();
    /* Transmit SCIFA9 data */
    R_SCIFA9_Serial_Send((uint8_t *)&g_scifa9_tx_buf, 1U);

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

r_cg_scifa_user.c

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

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

static void r_scifa9_callback_transmitend(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop the SCIFA9 channel */
    R_SCIFA9_Stop();
    /* End user code. Do not edit comment generated here */
}
### 3.2.27 I2C bus interface (RIIC)

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

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

Performs initialization necessary to control the I2C bus interface.

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

**[Syntax]**

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

**Remark** \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_RIICn_Create_UserInit**

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

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

**[Syntax]**

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

**Remark**  
`n` is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_riicn_error_interrupt**

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

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

**[Syntax]**

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

**Remark**  \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_riic\_receive\_interrupt**

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

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

**[Syntax]**

```c
static void    r_riic\_receive\_interrupt ( void );
```

**Remark** \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_riicn_transmit_interrupt

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

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

[Syntax]

static void r_riicn_transmit_interrupt ( void );

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.
**r\_riicn\_transmitend\_interrupt**

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

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

**[Syntax]**

```c
static void r\_riicn\_transmitend\_interrupt ( void );
```

**Remark** \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_RIICn_Start

Starts RIIC communication.

[Syntax]

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

Remark

\( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
**R_RIICn_Stop**

Ends RIIC communication.

**[Syntax]**

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

Remark  

\( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_RIICn_Master_Send

Starts RIIC master transmission.

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

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

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

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

Remark  \( n \) is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint16_t ( \text{adr} );</td>
<td>Slave address</td>
</tr>
<tr>
<td>I</td>
<td>uint8_t * const tx_buf;</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t \tx_num;</td>
<td>Total amount of data to send</td>
</tr>
</tbody>
</table>

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ERROR1</td>
<td>Bus busy</td>
</tr>
<tr>
<td>MD_ERROR2</td>
<td>Invalid argument ( \text{adr} ) specification</td>
</tr>
</tbody>
</table>
**R_RIICn_Master_Receive**

Starts RIIC master reception.

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

**Remark 2.** This API function internally calls \texttt{R_RIICn_StartCondition} to handle processing to start RIIC master reception.

**Remark 3.** When performing a RIIC master reception, \texttt{R_RIICn_Start} must be called before this API function is called.

**[Syntax]**

```c
#include "r_cg_macrodriver.h"

MD_STATUD R_RIICn_Master_Receive ( uint16_t adr, uint8_t * const rx_buf, uint16_t rx_num );
```

Remark \( n \) is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint16_t adr;</td>
<td>Slave address</td>
</tr>
<tr>
<td>O</td>
<td>uint8_t * const rx_buf;</td>
<td>Pointer to a buffer to store the reception data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t rx_num;</td>
<td>Total amount of data to receive</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ERROR1</td>
<td>Bus busy</td>
</tr>
<tr>
<td>MD_ERROR2</td>
<td>Invalid argument ( adr ) specification</td>
</tr>
</tbody>
</table>
**R_RIICn_Slave_Send**

Starts RIIC slave transmission.

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

Remark 2. When performing a RIIC slave transmission, `R_RIICn_Start` must be called before this API function is called.

**[Syntax]**

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

Remark  `n` is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint8_t * const tx_buf;</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t tx_num;</td>
<td>Total amount of data to send</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
</tbody>
</table>
**R_RIICn_Slave_Receive**

Starts RIIIC slave reception.

Remark 1. This API function performs byte-level RIIIC slave reception the number of times specified by the argument `rx_num` and stores the data in the buffer specified by the argument `rx_buf`.

Remark 2. When performing a RIIIC slave reception, `R_RIICn_Start` must be called before this API function is called.

**[Syntax]**

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

Remark \( n \) is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint8_t * const rx_buf</td>
<td>Pointer to a buffer to store the reception data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t rx_num</td>
<td>Total amount of data to receive</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
</tbody>
</table>
**R_RIICn_StartCondition**

Issues the start condition and causes a transfer error and an event generation interrupt (EEI).

**Remark 1.** This API function is called as the internal function of `R_RIICn_Master_Send` and `R_RIICn_Master_Receive`.

**Remark 2.** `r_riicn_error_interrupt` is called in response to calling of this API function.

**[Syntax]**

```c
void R_RIICn_StartCondition ( void );
```

Remark `n` is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_RIICn_StopCondition**

Issues the stop condition and causes a transfer error and an event generation interrupt (EEI).

Remark  
*r_riicn_error_interrupt* is called in response to calling of this API function.

**[Syntax]**

```
void R_RIICn_StopCondition ( void );
```

Remark  
*n* is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
Of the internal processing for transfer errors and event generation interrupts (EEI), this function handles processing specialized in the arbitration-lost detection, NACK detection, and timeout detection.

Remark This API function is called as the `r_riicn_error_interrupt` callback routine.

**[Syntax]**

```c
#include "r_cg_macrodriver.h"
static void r_riicn_callback_receiveerror ( MD_STATUS status );
```

Remark  $n$ is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>MD_STATUS status;</td>
<td>Source of the transfer errors and event generation interrupts&lt;br&gt;MD_ERROR1 : Arbitration-lost detection&lt;br&gt;MD_ERROR2 : Timeout detection&lt;br&gt;MD_ERROR3 : NACK detection</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
r_riicn_callback_transmitend

Of the internal processing for transfer errors and event generation interrupts (EEI), this function handles processing specialized in the start condition detection in response to calling of R_RIICn_Master_Send.

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

[Syntax]

static void r_riicn_callback_transmitend ( void );

Remark n is the channel number.

[Argument(s)]
None.

[Return value]
None.
Of the internal processing for transfer errors and event generation interrupts (EEI), this function handles processing specialized in the start condition detection in response to calling of `R_RIICn_Master_Receive`.

Remark This API function is called as the `r_riicn_error_interrupt` callback routine.

**Syntax**

```c
static void r_riicn_callback_receiveend ( void );
```

Remark \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
Usage example
Master transmission four times

[GUI setting example]

<table>
<thead>
<tr>
<th>I2C Bus Interface</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIIC2</td>
<td>Used</td>
</tr>
<tr>
<td>Function setting</td>
<td>I2C mode (Master)</td>
</tr>
<tr>
<td>SCL2</td>
<td>P16</td>
</tr>
<tr>
<td>SDA2</td>
<td>P17</td>
</tr>
<tr>
<td>I2CMaster2</td>
<td>Used</td>
</tr>
<tr>
<td>Baudrate</td>
<td>10 (kbps)</td>
</tr>
<tr>
<td>Noise filter stage</td>
<td>Single-stage filter</td>
</tr>
<tr>
<td>Enable SDA output delay</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable timeout function</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable master arbitration-lost detection</td>
<td>Used</td>
</tr>
<tr>
<td>Enable NACK transmission arbitration-lost detection</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable suspension during NACK reception</td>
<td>Used</td>
</tr>
<tr>
<td>TXI priority</td>
<td>Level 15 (highest)</td>
</tr>
<tr>
<td>RXI priority</td>
<td>Level 15 (highest)</td>
</tr>
<tr>
<td>TEI2, EEI2 priority (Group BL1)</td>
<td>Level 15 (highest)</td>
</tr>
<tr>
<td>Enable timeout interrupt (TMOI)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable arbitration-lost interrupt (ALI)</td>
<td>Used</td>
</tr>
<tr>
<td>Enable start condition detection interrupt (STI)</td>
<td>Used</td>
</tr>
<tr>
<td>Enable stop condition detection interrupt (SPI)</td>
<td>Used</td>
</tr>
<tr>
<td>Enable NACK reception interrupt (NAKI)</td>
<td>Used</td>
</tr>
<tr>
<td>Transfer end</td>
<td>Used</td>
</tr>
<tr>
<td>Receive end</td>
<td>Used</td>
</tr>
<tr>
<td>Receive error</td>
<td>Used</td>
</tr>
</tbody>
</table>

Interrupt Controller Unit

<table>
<thead>
<tr>
<th>ICU</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group</td>
<td>Used</td>
</tr>
<tr>
<td>Group BL1</td>
<td>Used</td>
</tr>
<tr>
<td>Group BL1 priority</td>
<td>Level 15 (highest)</td>
</tr>
</tbody>
</table>
[API setting example]

r_cg_main.c

/* Start user code for global. Do not edit comment generated here */
extern volatile uint8_t g_riic2_tx_buf[2];
/* End user code. Do not edit comment generated here */

void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start the RIIC2 Bus Interface */
    R_RIIC2_Start();

    /* Send RIIC2 data to slave device */
    R_RIIC2_Master_Send(0x00A0, (uint8_t *)g_riic2_tx_buf, 2U);

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

/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_riic2_tx_buf[2];
volatile uint8_t g_riic2_tx_cnt;
/* End user code. Do not edit comment generated here */

void R_RIIC2_Create_UserInit(void)
{
    /* Start user code. Do not edit comment generated here */
        g_riic2_tx_cnt = 0U;
        g_riic2_tx_buf[0] = g_riic2_tx_cnt;
        g_riic2_tx_buf[1] = 0x01;
    /* End user code. Do not edit comment generated here */
}

static void r_riic2_callback_transmitend(void)
{
    /* Start user code. Do not edit comment generated here */
    if (++g_riic2_tx_cnt < 4U)
    {
        g_riic2_tx_buf[0] = g_riic2_tx_cnt;
        g_riic2_tx_buf[1] += 0x01;
        /* Send RIIC2 data to slave device */
        R_RIIC2_Master_Send(0x00A0, (uint8_t *)g_riic2_tx_buf, 2U);
    }
    else
    {
        /* Stop the RIIC2 Bus Interface */
        R_RIIC2_Stop();
    }
    /* End user code. Do not edit comment generated here */
}
3.2.28 Serial peripheral interface (RSPI)

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

Table 3.28 API Functions: [Serial Peripheral Interface]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_RSPIn_Create</td>
<td>Performs initialization necessary to control the serial peripheral interface.</td>
</tr>
<tr>
<td>R_RSPIn_Create_UserInit</td>
<td>Performs user-defined initialization relating to the serial peripheral interface.</td>
</tr>
<tr>
<td>r_rspin_receive_interrupt</td>
<td>Performs processing in response to the receive buffer full interrupts.</td>
</tr>
<tr>
<td>r_rspin_transmit_interrupt</td>
<td>Performs processing in response to the transmit buffer error interrupts.</td>
</tr>
<tr>
<td>r_rspin_error_interrupt</td>
<td>Performs processing in response to the RSPI error interrupts.</td>
</tr>
<tr>
<td>r_rspin_idle_interrupt</td>
<td>Performs processing in response to the RSPI idle interrupts.</td>
</tr>
<tr>
<td>R_RSPIn_Start</td>
<td>Starts RSPI communication.</td>
</tr>
<tr>
<td>R_RSPIn_Stop</td>
<td>Ends RSPI communication.</td>
</tr>
<tr>
<td>R_RSPIn_Send</td>
<td>Starts RSPI transmission.</td>
</tr>
<tr>
<td>R_RSPIn_Send_Receive</td>
<td>Starts RSPI transmission/reception.</td>
</tr>
<tr>
<td>r_rspin_callback_receiveend</td>
<td>Performs processing in response to the receive buffer full interrupts.</td>
</tr>
<tr>
<td>r_rspin_callback_error</td>
<td>Performs processing in response to the RSPI error interrupts.</td>
</tr>
<tr>
<td>r_rspin_callback_transmitend</td>
<td>Performs processing in response to the RSPI idle interrupts.</td>
</tr>
</tbody>
</table>
R_RSPI\_n\_Create

Performs initialization necessary to control the serial peripheral interface.

Remark This API function is called from R\_Systeminit before main() is executed.

**[Syntax]**

```c
void R_RSPI\_n\_Create ( void );
```

Remark \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_RSPIn_Create_UserInit**

Performs user-defined initialization relating to the serial peripheral interface.

**Remark**
This API function is called as the R_RSPIn_Create callback routine.

**[Syntax]**

```c
void R_RSPIn_Create_UserInit ( void );
```

**Remark**

\( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_rspi_n_receive_interrupt

Performs processing in response to the receive buffer full interrupts.

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

[Syntax]

```c
static void    r_rspi_n_receive_interrupt ( void);
```

Remark $n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
r_rspin_transmit_interrupt

Performs processing in response to the transmit buffer empty interrupts.

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

[Syntax]

static void r_rspin_transmit_interrupt ( void );

Remark  \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
**r_rspin_error_interrupt**

Performs processing in response to the RSPI error interrupts.

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

**[Syntax]**

```c
static void r_rspin_error_interrupt ( void );
```

**Remark** \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_rspin_idle_interrupt**

Performs processing in response to the RSPI idle interrupts.

**Remark**  This API function is called to run interrupt processing for the RSPI idle interrupts.

**[Syntax]**

```c
static void r_rspin_idle_interrupt ( void );
```

**Remark**  \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
## R_RSPIn_Start

Starts RSPI communication.

### [Syntax]

```c
void R_RSPIn_Start ( void );
```

**Remark**  
$n$ is the channel number.

### [Argument(s)]

None.

### [Return value]

None.
R_RSPI\_n\_Stop

Ends RSPI communication.

**[Syntax]**

```c
void R_RSPI\_n\_Stop ( void );
```

Remark \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_RSPI\_n\_Send**

Starts RSPI transmission.

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

Remark 2. When performing a RSPI transmission, `R_RSPIn_Start` must be called before this API function is called.

**[Syntax]**

```c
#include "r_cg_macrodriver.h"
MD_STATUS R_RSPIn_Send ( uint32_t * const tx_buf, uint16_t tx_num );
```

Remark \( n \) is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint32_t * const tx_buf;</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t tx_num;</td>
<td>Total amount of data to send</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument ( tx_num ) specification</td>
</tr>
</tbody>
</table>
R_RSPIn_Send_Receive

Starts RSPI transmission/reception.

Remark 1. This API function repeats RSPI transmission in byte units the number of times specified by the argument \( tx\_num \) from the buffer at the location specified by the argument \( tx\_buf \).

Remark 2. This API function repeats RSPI reception processing in byte units the number of times specified by the argument \( tx\_num \) and then stores the received data in the buffer at the location specified by the argument \( rx\_buf \).

Remark 3. When performing a RSPI transmission/reception, \texttt{R_RSPIn_Start} must be called before this API function is called.

[Syntax]

```c
#include "r_cg_macrodriver.h"

MD_STATUS R_RSPIn_Send_Receive ( uint32_t * const tx_buf, uint16_t tx_num,
                                  uint32_t * const rx_buf );
```

Remark \( n \) is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint32_t * const tx_buf;</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t tx_num;</td>
<td>Total amount of data to send/receive</td>
</tr>
<tr>
<td>O</td>
<td>uint32_t * const rx_buf;</td>
<td>Pointer to a buffer to store the reception data</td>
</tr>
</tbody>
</table>

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument ( tx_num ) specification</td>
</tr>
</tbody>
</table>

r_rspi_callback_receiveend

Performs processing in response to the receive buffer full interrupts.

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

**[Syntax]**

```c
static void r_rspin_callback_receiveend ( void );
```

Remark \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
### r_rspin_callback_error

Performs processing in response to the RSPI error interrupts.

**Remark** This API function is called as the `r_rspin_error_interrupt` callback routine.

#### [Syntax]

```c
static void    r_rspin_callback_error ( uint8_t
err_type);
```

**Remark** \( n \) is the channel number.

#### [Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint8_t err_type;</td>
<td>Source of the RSPI error interrupt (x is undefined)</td>
</tr>
<tr>
<td></td>
<td>xxxx00x1B</td>
<td>: Overrun error detection</td>
</tr>
<tr>
<td></td>
<td>xxxx01x0B</td>
<td>: Mode fault error detection</td>
</tr>
<tr>
<td></td>
<td>xxxx10x0B</td>
<td>: Parity error detection</td>
</tr>
</tbody>
</table>

#### [Return value]

None.
**r_rspin_callback_transmitend**

Performs processing in response to the RSPI idle interrupts.

Remark. This API function is called as the `r_rspin_idle_interrupt` callback routine.

**[Syntax]**

```c
static void r_rspin_callback_transmitend ( void );
```

Remark  \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
Usage example
When the reception of the same number of data as the transmission is completed, the communication is end.

[GUI setting example]

<table>
<thead>
<tr>
<th>Serial Peripheral Interface</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSPi0</td>
<td>Used</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function setting</th>
<th>SPI operation (four-wire method)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Master transmit/receive</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RSPCKA pin</th>
<th>PC5 (A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOSIA pin</td>
<td>PC6 (A)</td>
</tr>
<tr>
<td>MISOA pin</td>
<td>PC7 (A)</td>
</tr>
<tr>
<td>Rspi4Wimetho_d_TransmitRecei_ve_Masterer0</td>
<td>Used</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Buffer size</th>
<th>64 bits (buffer accessed in words)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parity bit</td>
<td>Does not add the parity bit to transmit data and does not check the parity bit of receive data</td>
</tr>
<tr>
<td>Base bit rate</td>
<td>1000(kbps)(Actual value: 1000 Error: 0%)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Period from beginning of SSL signal assertion to RSPCK oscillation (RSPCK delay)</th>
<th>1 RSPCK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period from transmission of final RSPCK edge to negation of the SSL signal (SSL negation delay)</td>
<td>1 RSPCK</td>
</tr>
<tr>
<td>SSL signal non-active period after termination of a serial transfer (next-access delay)</td>
<td>1 RSPCK + 2 PCLK</td>
</tr>
<tr>
<td>MOSI idle value (MOSI output during SSL negation period in master mode)</td>
<td>Final data from previous transfer</td>
</tr>
<tr>
<td>SSLA0 pin</td>
<td>Unused</td>
</tr>
<tr>
<td>SSLA1 pin</td>
<td>PC0 (A) (Active low)</td>
</tr>
<tr>
<td>SSLA2 pin</td>
<td>Unused</td>
</tr>
<tr>
<td>SSLA3 pin</td>
<td>Unused</td>
</tr>
<tr>
<td>Output pin mode selection (RSPCK, SSL and MOSI on master mode / MISO on slave mode)</td>
<td>CMOS output</td>
</tr>
<tr>
<td>Loopback mode selection</td>
<td>Normal mode</td>
</tr>
<tr>
<td>Transmit data handling</td>
<td>Data handled in interrupt service routine0</td>
</tr>
<tr>
<td>SPRIO priority</td>
<td>Level 15 (highest)</td>
</tr>
<tr>
<td>SPEI0 priority</td>
<td>Level 15 (highest)</td>
</tr>
<tr>
<td>Enable error interrupt (SPEI0)</td>
<td>Used</td>
</tr>
<tr>
<td>SPEI0, SPI10 priority (Group AL0)</td>
<td>Level 15 (highest)</td>
</tr>
<tr>
<td>Transmission end</td>
<td>Used</td>
</tr>
<tr>
<td>Reception end</td>
<td>Used</td>
</tr>
<tr>
<td>Error detection</td>
<td>Used</td>
</tr>
<tr>
<td>Number of commands, number of frames</td>
<td>Number of commands: 1, number of transfer frames: 1</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>Command0;</td>
<td></td>
</tr>
<tr>
<td>Data length</td>
<td>8 bits</td>
</tr>
<tr>
<td>Format</td>
<td>MSB-first</td>
</tr>
<tr>
<td>RSPCK phase</td>
<td>Data variation on odd edge, data sampling on even edge</td>
</tr>
<tr>
<td>RSPCK polarity</td>
<td>Low when idle</td>
</tr>
<tr>
<td>Bit rate selection</td>
<td>Base bit rate / 8</td>
</tr>
<tr>
<td>SSL signal assertion</td>
<td>SSL0</td>
</tr>
<tr>
<td>SSL negation operation</td>
<td>Negates all SSL signals upon completion of transfer</td>
</tr>
<tr>
<td>RSPCK delay</td>
<td>1RSPCK</td>
</tr>
<tr>
<td>SSL negation delay</td>
<td>1RSPCK</td>
</tr>
<tr>
<td>Next-access delay</td>
<td>1RSPCK + 2CLK</td>
</tr>
</tbody>
</table>
[API setting example]

** r_cg_main.c

```c
/* Start user code for global. Do not edit comment generated here */
extern volatile uint32_t g_rspi0_tx_buf[4];
extern volatile uint32_t g_rspi0_rx_buf[4];
/* End user code. Do not edit comment generated here */

void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start the RSPI0 module operation */
    R_RSPI0_Start();

    /* Send and receive RSPI0 data */
    R_RSPI0_Send_Receive((uint32_t *)g_rspi0_tx_buf, 4U, (uint32_t *)g_rspi0_rx_buf);

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

** r_cg_rspi_user.c

```c
/* Start user code for global. Do not edit comment generated here */
volatile uint32_t g_rspi0_tx_buf[4];
volatile uint32_t g_rspi0_rx_buf[4];
/* End user code. Do not edit comment generated here */

void R_RSPI0_Create_UserInit(void)
{
    /* Start user code. Do not edit comment generated here */
    g_rspi0_tx_buf[0] = 0x000000FF;
    g_rspi0_tx_buf[1] = 0x0000FF00;
    g_rspi0_tx_buf[2] = 0x00FF0000;
    g_rspi0_tx_buf[3] = 0xFF000000;
    /* End user code. Do not edit comment generated here */
}

static void r_rspi0_callback_receiveend(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop the RSPI0 module operation */
    R_RSPI0_Stop();
    /* End user code. Do not edit comment generated here */
}
3.2.29 CRC calculator (CRC)

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

Table 3.29 CRC API Functions: [CRC calculator]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_CRC_SetCRC8</td>
<td>Initializes the CRC calculator for 8-bit CRC calculation (CRC generating polynomial: (X^8 + X^2 + X + 1)).</td>
</tr>
<tr>
<td>R_CRC_SetCRC16</td>
<td>Initializes the CRC calculator for 16-bit CRC calculation (CRC generating polynomial: (X^{16} + X^{15} + X^2 + 1)).</td>
</tr>
<tr>
<td>R_CRC_SetCCITT</td>
<td>Initializes the CRC calculator for 16-bit CRC calculation (CRC generating polynomial: (X^{16} + X^{12} + X^3 + 1)).</td>
</tr>
<tr>
<td>R_CRC_SetCRC32</td>
<td>Initializes the CRC calculator for 32-bit CRC calculation (CRC generating polynomial: (X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^6 + X^2 + X + 1)).</td>
</tr>
<tr>
<td>R_CRC_SetCRC32C</td>
<td>Initializes the CRC calculator for 32-bit CRC calculation (CRC generating polynomial: (X^{32} + X^{28} + X^{27} + X^{26} + X^{25} + X^{23} + X^{22} + X^{20} + X^{19} + X^{18} + X^{14} + X^{13} + X^{11} + X^{10} + X^9 + X^8 + X^6 + 1)).</td>
</tr>
<tr>
<td>R_CRC_Input_Data</td>
<td>Sets the initial value of the data from which the CRC is to be calculated.</td>
</tr>
<tr>
<td>R_CRC_Get_Result</td>
<td>Gets the result of operation.</td>
</tr>
</tbody>
</table>
**R_CRC_SetCRC8**

Initializes the CRC calculator for 8-bit CRC calculation (CRC generating polynomial: $X^8 + X^2 + X + 1$).

**[Syntax]**

RX65N/RX651

```c
void R_CRC_SetCRC8 ( void );
```

Other devices

```c
#include "r_cg_crc.h"
void R_CRC_SetCRC8 ( crc_bitorder order );
```

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>crc_bitorder</td>
<td>CRC calculation switching type</td>
</tr>
<tr>
<td></td>
<td>order</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CRC_LSB</td>
<td>: LSB-first</td>
</tr>
<tr>
<td></td>
<td>CRC_MSB</td>
<td>: MSB-first</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
R_CRC_SetCRC16

Initializes the CRC calculator for 16-bit CRC calculation (CRC generating polynomial: $X^{16} + X^{15} + X^2 + 1$).

[Syntax]
RX65N/RX651
void R_CRC_SetCRC16 ( void );

Other devices
#include "r_cg_crc.h"
void R_CRC_SetCRC16 ( crc_bitorder order );

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>crc_bitorder</td>
<td>CRC calculation</td>
</tr>
</tbody>
</table>

order

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC_LSB</td>
<td>LSB-first</td>
</tr>
<tr>
<td>CRC_MSB</td>
<td>MSB-first</td>
</tr>
</tbody>
</table>

[Return value]
None.
R_CRC_SetCCITT

Initializes the CRC calculator for the 16-bit CRC calculation (CRC generating polynomial: \(X^{16} + X^{12} + X^5 + 1\)).

[Syntax]
RX65N/RX651

```c
void    R_CRC_SetCCITT ( void );
```

Other devices

```c
#include       "r_cg_crc.h"
void    R_CRC_SetCCITT ( crc_bitorder order );
```

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>crc_bitorder order;</td>
<td>CRC calculation switching type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CRC_LSB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>: LSB-first</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CRC_MSB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>: MSB-first</td>
</tr>
</tbody>
</table>

[Return value]
None.
R_CRC_SetCRC32

Initializes the CRC calculator for the 32-bit CRC calculation (CRC generating polynomial: $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^{8} + X^{7} + X^{5} + X + 1$).

[Syntax]

```c
void R_CRC_SetCRC32 ( void );
```

[Argument(s)]

None.

[Return value]

None.
**R_CRC_SetCRC32C**

Initializes the CRC calculator for the 32-bit CRC calculation (CRC generating polynomial: \( X^{32} + X^{28} + X^{27} + X^{26} + X^{25} + X^{23} + X^{22} + X^{20} + X^{19} + X^{18} + X^{14} + X^{13} + X^{11} + X^{10} + X^9 + X^8 + X^6 + 1 \)).

**Syntax**

```c
void R_CRC_SetCRC32C ( void );
```

**Argument(s)**

None.

**Return value**

None.
**R_CRC_Input_Data**

Sets the initial value of the data from which the CRC is to be calculated.

**[Syntax]**

```c
void    R_CRC_Input_Data ( uint8_t  data );
```

```c
void    R_CRC_Input_Data ( uint32_t  data );
```

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint8_t data;</td>
<td>The initial value of the data from which the CRC is to be calculated</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint32_t data;</td>
<td>The initial value of the data from which the CRC is to be calculated</td>
</tr>
</tbody>
</table>

**Remark**  Argument size differs for each device group.

**[Return value]**

None.
R_CRC_Get_Result

Gets the result of operation.

[Syntax]

```c
void R_CRC_Get_Result ( uint16_t * const result );

void R_CRC_Get_Result ( uint32_t * const result );
```

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint16_t * const</td>
<td>Pointer to the location where the result of operation is stored</td>
</tr>
<tr>
<td>O</td>
<td>uint32_t * const</td>
<td>Pointer to the location where the result of operation is stored</td>
</tr>
</tbody>
</table>

Remark  Argument size differs for each device group.

[Return value]

None.
**Usage example**

Generates CRC code and adds it to transmission data.
A CRC code is generated from the analyzed received data, and it is checked whether the received data is correct.

**[GUI setting example]**

<table>
<thead>
<tr>
<th>CRC Calculator</th>
<th>Used</th>
<th>Unused</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>Generate functions for CRC-8 calculation</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>Generate functions for CRC-16 calculation</td>
<td>Unused</td>
<td></td>
</tr>
<tr>
<td>Generate functions for CRC-CCITT calculation</td>
<td>Unused</td>
<td></td>
</tr>
<tr>
<td>Initial value</td>
<td>0x0000</td>
<td></td>
</tr>
<tr>
<td>Invert result of calculated value</td>
<td>Unused</td>
<td></td>
</tr>
</tbody>
</table>

**[API setting example]**

```c
/* Start user code for adding. Do not edit comment generated here */

volatile uint8_t tx_buf[2];
volatile uint16_t result;

void tx_func(void)
{
    /* Set CRC module using CRC8 algorithm */
    R_CRC_SetCRC8(CRC_LSB);

    /* Restore transmit data */
    tx_buf[0] = 0xF0;

    /* Write data to CRC input register */
    R_CRC_Input_Data(tx_buf[0]);

    /* Get result from CRC output register */
    R_CRC_Get_Result((uint16_t *)&result);

    /* Restore CRC code */
    tx_buf[1] = (uint8_t)(result);

    /* Transmit "tx_buf" */
}
/* End user code. Do not edit comment generated here */```
rx_func.c

/* Start user code for adding. Do not edit comment generated here */
volatile uint8_t rx_buf[2];
volatile uint16_t result;
volatile uint8_t err_f;

void rx_func(void)
{
    /* Clear error flag */
    err_f = 0U;

    /*** Receive (Restore the receive data in "rx_buf") ***/

    /* Set CRC module using CRC8 algorithm */
    R_CRC_SetCRC8(CRC_LSB);

    /* Write data to CRC input register */
    R_CRC_Input_Data(rx_buf[0]);

    /* Get result from CRC output register */
    R_CRC_Get_Result((uint16_t *)&result);

    /* Check the receive data */
    if (rx_buf[1] != (uint8_t)(result))
    {
        /* Set error flag */
        err_f = 1U;
    }
}

/* End user code. Do not edit comment generated here */
3.2.30 12-bit A/D converter (S12AD)

Below is a list of API functions output by the Code Generator for 12-bit A/D converter use.

Table 3.30 API Functions: [12-Bit A/D Converter]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_S12ADn_Create</td>
<td>Performs initialization necessary to control the 12-bit A/D converter.</td>
</tr>
<tr>
<td>R_S12ADn_Create_UserInit</td>
<td>Performs user-defined initialization relating to the 12-bit A/D converter.</td>
</tr>
<tr>
<td>r_s12adn_interrupt</td>
<td>Performs processing in response to the A/D scan end interrupt.</td>
</tr>
<tr>
<td>r_s12adn_groupb_interrupt</td>
<td>Performs processing in response to the group B scan end interrupt.</td>
</tr>
<tr>
<td>R_S12ADn_Start</td>
<td>Starts A/D conversion.</td>
</tr>
<tr>
<td>R_S12ADn_Stop</td>
<td>Ends A/D conversion.</td>
</tr>
<tr>
<td>R_S12ADn_Get_ValueResult</td>
<td>Gets the result of conversion.</td>
</tr>
<tr>
<td>R_S12ADn_Set_CompareValue</td>
<td>Sets compare level.</td>
</tr>
<tr>
<td>r_s12adn_compare_interrupt</td>
<td>Performs processing in response to the compare interrupt.</td>
</tr>
</tbody>
</table>
R_S12ADn_Create

Performs initialization necessary to control the 12-bit A/D converter.

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

[Syntax]

```
void R_S12ADn_Create ( void );
```

Remark  \( n \) is the unit number.

[Argument(s)]

None.

[Return value]

None.
R_S12ADn_Create_UserInit

Performs user-defined initialization relating to the 12-bit A/D converter.

Remark This API function is called as the R_S12ADn_Create callback routine.

[Syntax]

```c
void R_S12ADn_Create_UserInit ( void );
```

Remark $n$ is the unit number.

[Argument(s)]

None.

[Return value]

None.
**r_s12adn_interrupt**

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

**Remark**  This API function is called to run interrupt processing for the A/D scan end interrupt, which is generated on completion of scanning of the analog inputs.

**[Syntax]**

```c
static void r_s12adn_interrupt ( void );
```

**Remark**  `n` is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_s12adn_groupb_interrupt**

Performs processing in response to the group B scan end interrupt.

**Remark** This function is called to run interrupt processing for the group B scan end interrupt, which is generated when scanning of the analog inputs allocated to group B is completed.

**[Syntax]**

```c
static void r_s12adn_groupb_interrupt ( void );
```

**Remark** $n$ is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_S12ADn_Start

Starts A/D conversion.

[Syntax]

```c
void R_S12ADn_Start ( void );
```

Remark

*n* is the unit number.

[Argument(s)]

None.

[Return value]

None.
**R_S12ADn_Stop**

Ends A/D conversion.

**[Syntax]**

```c
void R_S12ADn_Stop ( void );
```

*Remark*  
\( n \) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_S12ADn_Get_ValueResult

Gets the result of conversion.

[Syntax]

```c
#include       "r_cg_s12ad.h"
void    R_S12ADn_Get_ValueResult ( ad_channel_t channel, uint16_t * const buffer );
```

Remark  

n is the unit number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ad_channel_t channel;</td>
<td>Channel number</td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL0 : Input channel AN000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL1 : Input channel AN001</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL2 : Input channel AN002</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL3 : Input channel AN003</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL4 : Input channel AN004</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL5 : Input channel AN005</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL6 : Input channel AN006</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL7 : Input channel AN007</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL8 : Input channel AN008</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL9 : Input channel AN009</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL10 : Input channel AN010</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL11 : Input channel AN011</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL12 : Input channel AN012</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL13 : Input channel AN013</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL14 : Input channel AN014</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL15 : Input channel AN015</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL16 : Input channel AN016</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL17 : Input channel AN017</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL18 : Input channel AN018</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL19 : Input channel AN019</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL20 : Input channel AN020</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADTEMPSENSOR : Extended analog input (temperature sensor output)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADINTERREFVOLT : Extended analog input (internal reference voltage)</td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>uint16_t * const buffer;</td>
<td>Pointer to the area where the results of conversion are stored</td>
</tr>
</tbody>
</table>

[Return value]

None.
R_S12ADn_Set_CompareValue

Sets compare level.

[Syntax]

```c
void R_S12ADn_Set_CompareValue ( uint16_t reg_value0, uint16_t reg_value1);
```

Remark  

$n$ is the unit number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint16_t reg_value0;</td>
<td>Register value set to the compare relel register 0</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t reg_value1;</td>
<td>Register value set to the compare relel register 1</td>
</tr>
</tbody>
</table>

[Return value]

None.
**r_s12adn_compare_interrupt**

Performs processing in response to the compare interrupt.

**Remark**
This API function is called to run the interrupt processing for the compare interrupt.

**[Syntax]**

```c
static void r_s12adn_compare_interrupt ( void );
```

**Remark**

\[n\] is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
Usage example

Get the A/D conversion result.

[GUI setting example]

<table>
<thead>
<tr>
<th>12-Bit A/D Converter</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>S12AD0</td>
<td>Used</td>
</tr>
<tr>
<td>AnalogInPutChan nelMode 0</td>
<td>Used</td>
</tr>
<tr>
<td>S12AD0 operation setting</td>
<td>Used</td>
</tr>
<tr>
<td>Operation mode setting</td>
<td>Continuous scan mode</td>
</tr>
<tr>
<td>Self diagnosis setting</td>
<td>Unused</td>
</tr>
<tr>
<td>Disconnection detection assist setting</td>
<td>Unused</td>
</tr>
<tr>
<td>A / D conversion value count setting</td>
<td>Addition mode</td>
</tr>
<tr>
<td>Analog input channel setting</td>
<td></td>
</tr>
<tr>
<td>AN000 Convert (Group A)</td>
<td>Used</td>
</tr>
<tr>
<td>AN000 Add/Average AD value</td>
<td>Unused</td>
</tr>
<tr>
<td>AN000 Dedicated sample and hold</td>
<td>Unused</td>
</tr>
<tr>
<td>Conversion start trigger (Group A)</td>
<td>Software trigger</td>
</tr>
<tr>
<td>Data placement</td>
<td>Right-alignment</td>
</tr>
<tr>
<td>Automatic clearing</td>
<td>Disable automatic clearing</td>
</tr>
<tr>
<td>Data accuracy</td>
<td>12-bit accuracy</td>
</tr>
<tr>
<td>AN000 Input sampling time</td>
<td>3.667(us)</td>
</tr>
<tr>
<td>Total conversion time (Group A)</td>
<td>7.167(us)</td>
</tr>
<tr>
<td>Enable AD conversion end interrupt (S12ADI0)</td>
<td>Used</td>
</tr>
<tr>
<td>S12ADI0 priority</td>
<td>Level 15 (highest)</td>
</tr>
<tr>
<td>Window function setting</td>
<td>Unused</td>
</tr>
<tr>
<td>Initial reference data 0 for comparison</td>
<td>0</td>
</tr>
<tr>
<td>Use comparator for AN000</td>
<td>Unused</td>
</tr>
</tbody>
</table>
[API setting example]

r_cg_main.c

```c
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start the AD0 converter */
    R_S12AD0_Start();

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

r_cg_s12ad_user.c

```c
/* Start user code for global. Do not edit comment generated here */
volatile uint16_t g_s12ad0_ch000_value;
/* End user code. Do not edit comment generated here */

static void r_s12ad0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Get result from the AD0 channel 0 (AN000) converter */
    R_S12AD0_Get_ValueResult(ADCHANNEL0, (uint16_t *)&g_s12ad0_ch000_value);
    /* End user code. Do not edit comment generated here */
}
```
3.2.31 D/A converter (DA)

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

Table 3.31 API Functions: [D/A Converter]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_DA_Create</td>
<td>Performs initialization necessary to control the D/A converter.</td>
</tr>
<tr>
<td>R_DA_Create_UserInit</td>
<td>Performs user-defined initialization relating to the D/A converter.</td>
</tr>
<tr>
<td>R_DAm_Start</td>
<td>Starts D/A conversion.</td>
</tr>
<tr>
<td>R_DAm_Stop</td>
<td>Ends D/A conversion.</td>
</tr>
<tr>
<td>R_DAm_Set_ConversionValue</td>
<td>Sets the data for D/A conversion.</td>
</tr>
</tbody>
</table>
R_DA_Create

Performs initialization necessary to control the D/A converter.

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

[Syntax]

```c
void R_DA_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.
R_DA_Create_UserInit

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

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

[Syntax]

void R_DA_Create_UserInit ( void );

[Argument(s)]

None.

[Return value]

None.
**R_DAm_Start**

Starts D/A conversion.

**[Syntax]**

```c
void R_DAm_Start ( void );
```

*Remark*  
$m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_DAm_Stop**

Ends D/A conversion.

**[Syntax]**

```c
void R_DAm_Stop ( void );
```

**Remark**  
$m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
Rputy143243EJ0100  Rev.1.00  Page 388 of 442
July 01, 2018

R_DA_m_Sett_ConversionValue

Sets the data for D/A conversion.

[Syntax]

```c
void R_DA_m_Sett_ConversionValue ( uint16_t reg_value );
```

Remark  \( m \) is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint16_t</td>
<td>( reg_value )</td>
</tr>
</tbody>
</table>

Data for D/A conversion

[Return value]

None.
Usage example

Start D / A conversion of channels 0 and 1.

[GUI setting example]

<table>
<thead>
<tr>
<th>D/A Converter</th>
<th>DA</th>
<th>D/A converter operation setting</th>
<th>Use DA0</th>
<th>Use DA1</th>
<th>Data format</th>
<th>D/A-A/D synchronous setting</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Used</td>
<td>Used</td>
<td>Right-alignment</td>
<td>Unused</td>
</tr>
</tbody>
</table>

[API setting example]

r_cg_main.c

```c
void main(void) {
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Set the DA0 converter value */
    R_DA0_Set_ConversionValue(0100U);
    /* Set the DA1 converter value */
    R_DA1_Set_ConversionValue(0200U);
    /* Enable the DA0 converter */
    R_DA0_Start();
    /* Enable the DA1 converter */
    R_DA1_Start();
    while (1U) {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```
3.2.32 12-bit D/A converter (R12DA)

Below is a list of API functions output by the Code Generator for 12-bit D/A converter use.

Table 3.32 API Functions: [12-Bit D/A Converter]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_R12DA_Create</td>
<td>Performs initialization necessary to control the 12-bit D/A converter.</td>
</tr>
<tr>
<td>R_R12DAn_Start</td>
<td>Starts D/A conversion.</td>
</tr>
<tr>
<td>R_R12DAn_Stop</td>
<td>Ends D/A conversion.</td>
</tr>
<tr>
<td>R_R12DAn_Set_ConversionValue</td>
<td>Sets the data for D/A conversion.</td>
</tr>
<tr>
<td>R_R12DA.Sync_Start</td>
<td>Starts synchronous D/A conversion.</td>
</tr>
<tr>
<td>R_R12DA.Sync_Stop</td>
<td>Ends synchronous D/A conversion.</td>
</tr>
<tr>
<td>R_R12DA.Create_UserInit</td>
<td>Performs user-defined initialization relating to the 12-bit D/A converter.</td>
</tr>
</tbody>
</table>
**R_R12DA_Create**

Performs initialization necessary to control the 12-bit D/A converter.

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

**[Syntax]**

```c
void    R_R12DA_Create ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_R12DA\_n\_Start

Starts 12-bit D/A conversion.

[Syntax]

\begin{verbatim}
void R_R12DA\_n\_Start ( void );
\end{verbatim}

Remark \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
**R_R12DA\_n\_Stop**

Ends 12-bit D/A conversion.

**[Syntax]**

```
void R_R12DA\_n\_Stop ( void );
```

Remark \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
### R_R12DA\_n\_Set\_ConversionValue

Sets the data for 12-bit D/A conversion.

#### Syntax

```
void R_R12DA\_n\_Set\_ConversionValue ( uint16\_t reg\_value );
```

**Remark**  
\( n \) is the channel number.

#### Argument(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint16_t reg_value;</td>
<td>Data for 12-bit D/A conversion</td>
</tr>
</tbody>
</table>

#### Return value

None.
R_R12DA_Sync_Start

Starts synchronous 12-bit D/A conversion.

[Syntax]
void R_R12DA_Sync_Start ( void );

[Argument(s)]
None.

[Return value]
None.
R_R12DA_Sync_Stop

Ends synchronous 12-bit D/A conversion.

[Syntax]

```c
void R_R12DA_Sync_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.
R_R12DA_Create_UserInit

Performs user-defined initialization relating to the 12-bit D/A converter.

Remark   This API function is called as the R_R12DA_Create callback routine.

[Syntax]

```c
void R_R12DA_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.
Usage example
Start D / A conversion of channels 0 and 1.

[GUI setting example]

<table>
<thead>
<tr>
<th>D/A Converter</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>DA</td>
<td>Used</td>
</tr>
<tr>
<td>D/A converter operation setting</td>
<td>Used</td>
</tr>
<tr>
<td>Use DA0</td>
<td>Used</td>
</tr>
<tr>
<td>Use DA1</td>
<td>Used</td>
</tr>
<tr>
<td>Data format</td>
<td>Right-alignment</td>
</tr>
<tr>
<td>D/A-A/D synchronous setting</td>
<td>Unused</td>
</tr>
</tbody>
</table>

(API setting example)

`r_cg_main.c`

```c
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Set the DA0 converter value */
    R_DA0_Set_ConversionValue(0100U);

    /* Set the DA1 converter value */
    R_DA1_Set_ConversionValue(0200U);

    /* Enable the DA0 converter */
    R_DA0_Start();

    /* Enable the DA1 converter */
    R_DA1_Start();

    while (1U)
    {
    
    }

    /* End user code. Do not edit comment generated here */
}
```
3.2.33 Comparator B (CMPB)

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

Table 3.33 API Functions: [Comparator B]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_CMPB_Create</td>
<td>Performs initialization necessary to control the Comparator B.</td>
</tr>
<tr>
<td>R_CMPB_Create_UserInit</td>
<td>Performs user-defined initialization relating to the Comparator B.</td>
</tr>
<tr>
<td>r_cmpb_cmpbn_interrupt</td>
<td>Performs processing in response to the comparator B interrupt.</td>
</tr>
<tr>
<td>R_CMPBn_Start</td>
<td>Starts comparison for analog input voltage.</td>
</tr>
<tr>
<td>R_CMPBn_Stop</td>
<td>Ends comparison for analog input voltage.</td>
</tr>
</tbody>
</table>
R_CMPB_Create

Performs initialization necessary to control the Comparator B.

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

[Syntax]

```c
void R_CMPB_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.
R_CMPB_Create_UserInit

Performs user-defined initialization relating to the Comparator B.

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

[Syntax]

```c
void    R_CMPB_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.
Performs processing in response to the comparator B interrupt.

**Remark** This API function is called to run interrupt processing for the comparator \( B_n \) interrupt, which is generated when the comparison result changes at this time.

**[Syntax]**

```c
static void r_cmpb_cmpbn_interrupt ( void );
```

**Remark** \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
\textbf{R_CMPB\textsubscript{n} Start}

Starts comparison for analog input voltage.

\textbf{[Syntax]}

\begin{verbatim}
void R_CMPB\textsubscript{n} Start ( void );
\end{verbatim}

Remark \textit{n} is the channel number.

\textbf{[Argument(s)]}

None.

\textbf{[Return value]}

None.
R_CMPBn_Stop

Ends comparison for analog input voltage.

[Syntax]

```c
void R_CMPBn_Stop ( void );
```

Remark  

$n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
Usage example

Set the flag to notify changing the comparison result.

[GUI setting example]

<table>
<thead>
<tr>
<th>Comparator B</th>
<th>CMPB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparator B0</td>
<td>Used</td>
</tr>
<tr>
<td>Comparator B1</td>
<td>Unused</td>
</tr>
<tr>
<td>Comparator B2</td>
<td>Unused</td>
</tr>
<tr>
<td>Comparator B3</td>
<td>Unused</td>
</tr>
<tr>
<td>Comparator B0, B1 speed setting</td>
<td>Low speed</td>
</tr>
<tr>
<td>Comparator B2, B3 speed setting</td>
<td>Low speed</td>
</tr>
<tr>
<td>Comparator B0 Mode setting</td>
<td>Normal</td>
</tr>
<tr>
<td>Comparator B0 Reference voltage setting</td>
<td>CVREFB0 input</td>
</tr>
<tr>
<td>Comparator b0 Enable digital filter</td>
<td>Unused</td>
</tr>
<tr>
<td>Comparator B0 Enable output (CMPOB0)</td>
<td>Unused</td>
</tr>
<tr>
<td>Enable comparator B0 interrupt (CMPB0)</td>
<td>Used</td>
</tr>
<tr>
<td>Comparator B0 Edge select</td>
<td>Rising</td>
</tr>
<tr>
<td>Comparator B0 priority</td>
<td>Level 15 (highest)</td>
</tr>
</tbody>
</table>
[API setting example]

r_cg_main.c

void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start comparator B0 */
    R_CMPB0_Start();

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

r_cg_cmpb_user.c

/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_cmpb0_f;
/* End user code. Do not edit comment generated here */

void R_CMPB_Create_UserInit(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Clear the flag */
    g_cmpb0_f = 0U;
    /* End user code. Do not edit comment generated here */
}

static void r_cmpb_cmpb0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Set the flag */
    g_cmpb0_f = 1U;
    /* End user code. Do not edit comment generated here */
}
3.2.34 Data operation circuit (DOC)

Below is a list of API functions output by the Code Generator for data operation circuit.

Table 3.34 API Functions: [Data Operation Circuit]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_DOC_Create</td>
<td>Performs initialization necessary to control the data operation circuit.</td>
</tr>
<tr>
<td>R_DOC_Create_UserInit</td>
<td>Performs user-defined initialization relating to the data operation circuit.</td>
</tr>
<tr>
<td>r_doc_dopcf_interrupt</td>
<td>Performs processing in response to the data operation circuit interrupt.</td>
</tr>
<tr>
<td>R_DOC_SetMode</td>
<td>Sets the operating mode and the initial value of the reference value for use by the data operation circuit.</td>
</tr>
<tr>
<td>R_DOC_WriteData</td>
<td>Sets the input value (value for comparison with, addition to, or subtraction from the reference value) for use by the data operation circuit.</td>
</tr>
<tr>
<td>R_DOC_GetResult</td>
<td>Gets the result of operation.</td>
</tr>
<tr>
<td>R_DOC_ClearFlag</td>
<td>Clears the data operation circuit flag.</td>
</tr>
</tbody>
</table>
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]

```c
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]**

```c
void R_DOC_Create_UserInit ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
**r_doc_dopcf_interrupt**

Performs processing in response to the data operation circuit interrupt.

**Remark**  
This API function is called to run interrupt processing for the data operation circuit interrupt, which is generated when the result of data comparison satisfies the condition for detection, the result of addition is greater than 0xFFFF, or the result of subtraction is less than 0x00.

**[Syntax]**

```c
static void r_doc_dopcf_interrupt ( void );
```

**Remark**  
API function name differs for each device group.

**[Argument(s)]**

None.

**[Return value]**

None.
R_DOC_SetMode

Sets the operating mode and the initial value of the reference value for use by the data operation circuit.

Remark 1. When COMPARE_MISMATCH or COMPARE_MATCH (data comparison mode) is specified as the mode of operation, the 16-bit reference value is stored in the DOC data setting register (DODSR).

Remark 2. When ADDITION (data addition mode) or SUBTRACTION (data subtraction mode) is specified for the mode (operation mode), the 16-bit value is stored in the DOC data setting register (DODSR) as the initial value.

[Syntax]

```c
#include      "r_cg_doc.h"
void    R_DOC_SetMode ( doc_mode_t mode, uint16_t value );
```

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>doc_mode_t mode;</td>
<td>Operating modes (including the condition for detection)</td>
</tr>
<tr>
<td></td>
<td>COMPARE_MISMATCH : Data comparison mode (mismatch)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMPARE_MATCH : Data comparison mode (match)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADDITION : Data addition mode</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SUBTRACTION : Data subtraction mode</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>uint16_t value;</td>
<td>Initial value of the reference value for use by the DOC</td>
</tr>
</tbody>
</table>

[Return value]

None.
**R_DOC_WriteData**

Sets the value for comparison with, addition to, or subtraction from the reference value.

**[Syntax]**

```c
void R_DOC_WriteData ( uint16_t data );
```

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint16_t data;</td>
<td>Input data for use in operation</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
R_DOC_GetResult

 Gets the result of operation.

[Syntax]

```c
void R_DOC_GetResult ( uint16_t * const data );
```

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint16_t * const data;</td>
<td>Pointer to the location where the result of operation is to be stored</td>
</tr>
</tbody>
</table>

[Return value]

None.
R_DOC_ClearFlag

Clears the data operation circuit flag.

[Syntax]

```c
void R_DOC_ClearFlag ( void );
```

[Argument(s)]

None.

[Return value]

None.
Usage example

Adds the array data and when it becomes larger than "FFFFh" gets the addition result by interrupt. (Data addition mode)

Subsequently, the mode is changed to the Data comparison mode (mismatch), and an interrupt is generated when anything other than "000h" is detected in the array data.

[GUI settings]

<table>
<thead>
<tr>
<th>Data Operation Circuit</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOC</td>
<td>Used</td>
</tr>
<tr>
<td>Generate function for setting DOC operation mode</td>
<td>Used</td>
</tr>
<tr>
<td>Operation mode</td>
<td>Data addition mode</td>
</tr>
<tr>
<td>Comparison reference/ Initial value of addition or subtraction result</td>
<td>0</td>
</tr>
<tr>
<td>Enable data operation circuit interrupt (DOPCF)</td>
<td>Used</td>
</tr>
<tr>
<td>DOPCF Priority (Group BL0)</td>
<td>Level 15 (highest)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interrupt Controller Unit</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICU</td>
<td>Used</td>
</tr>
<tr>
<td>Group</td>
<td>Used</td>
</tr>
<tr>
<td>Group BL0</td>
<td>Used</td>
</tr>
<tr>
<td>Group BL0 Priority</td>
<td>Level 15 (highest)</td>
</tr>
</tbody>
</table>
[API setting example]

r_cg_main.c

/* Start user code for global. Do not edit comment generated here */
extern volatile uint16_t data[16];
volatile uint8_t cnt;
/* End user code. Do not edit comment generated here */

void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    while (1U)
    {
        for (cnt = 0; cnt < 16U; cnt++)
        {
            /* Write new data to compare */
            R_DOC_WriteData(data[cnt]);
        }
    /* End user code. Do not edit comment generated here */
}

r_cg_doc_user.c

/* Start user code for global. Do not edit comment generated here */
volatile uint16_t data[16];
volatile uint16_t result;
/* End user code. Do not edit comment generated here */

void r_doc_dopc_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Get result */
    R_DOC_GetResult((uint16_t *)&result);
    /* Configure the operation mode of DOC */
    R_DOC_SetMode(COMPARE_MISMATCH, 0x0000);
    /* Clear DOPCI flag */
    R_DOC_ClearFlag();
    /* End user code. Do not edit comment generated here */
}
3.2.35 Low power timer (LPT)

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

Table 3.35 API Functions: [Low power timer]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_LPT_Create</td>
<td>Performs initialization necessary to control the low power timer.</td>
</tr>
<tr>
<td>R_LPT_Create_UserInit</td>
<td>Performs user-defined initialization relating to the low power timer.</td>
</tr>
<tr>
<td>R_LPT_Start</td>
<td>Starts counting by a low power timer.</td>
</tr>
<tr>
<td>R_LPT_Stop</td>
<td>Ends counting by a low power timer.</td>
</tr>
</tbody>
</table>
**R_LPT_Create**

Performs initialization necessary to control the low power timer.

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

**[Syntax]**

```c
void R_LPT_Create ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
**R_LPT_Create_UserInit**

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

**Remark** This API function is called as the R_LPT_Create callback routine.

**[Syntax]**

```c
void R_LPT_Create_UserInit ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_LPT_Start

Starts counting by a low power timer.

[Syntax]

```c
void R_LPT_Start ( void );
```

[Argument(s)]
None.

[Return value]
None.
R_LPT_Stop

Ends counting by a low power timer.

[Syntax]

```c
void R_LPT_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.
Usage example

Return from software standby mode to normal operation mode by LPT compare match via event link controller (ELC).

**[GUI settings]**

<table>
<thead>
<tr>
<th>Low Power Timer</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPT</td>
<td>Used</td>
</tr>
<tr>
<td>Low power timer operation setting</td>
<td>Used</td>
</tr>
<tr>
<td>Timer cycle value</td>
<td>8000ms (Actual value: 8000 Error: 0%)</td>
</tr>
<tr>
<td>Register value (LPTPRD)</td>
<td>59999</td>
</tr>
<tr>
<td>Compare value</td>
<td>29999</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Low Power Consumption</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPC</td>
<td>Used</td>
</tr>
<tr>
<td>LPC operation setting</td>
<td>Used</td>
</tr>
<tr>
<td>Initial operating power control mode</td>
<td>High-speed operating mode</td>
</tr>
<tr>
<td>Output of the Address bus and bus control signals</td>
<td>Retain the output state in software standby mode</td>
</tr>
<tr>
<td>Initial sleep mode return clock source</td>
<td>Disabled</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Event Link Controller</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELC</td>
<td>Used</td>
</tr>
<tr>
<td>ELC InterruptLP T</td>
<td>Used</td>
</tr>
<tr>
<td>LPT dedicated interrupt</td>
<td>Used</td>
</tr>
<tr>
<td>Event signal</td>
<td>LPT compare match</td>
</tr>
<tr>
<td>Operation on event</td>
<td>Issues an event to the CPU to wake up from CPU software (highest)</td>
</tr>
<tr>
<td>ELSR8I Priority</td>
<td>Level 15</td>
</tr>
</tbody>
</table>
r_cg_main.c

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

    /* Start LPT module */
    R_LPT_Start();

    /* Enable software standby mode */
    R_LPC_SoftwareStandby();

    /* Stop LPT module */
    R_LPT_Stop();

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

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_CMP_C_Create</td>
<td>Performs initialization necessary to control the Comparator C.</td>
</tr>
<tr>
<td>R_CMP_C_Create_UserInit</td>
<td>Performs user-defined initialization relating to the Comparator C.</td>
</tr>
<tr>
<td>r_cmpe_cmpcn_interrupt</td>
<td>Performs processing in response to the comparator C interrupt.</td>
</tr>
<tr>
<td>R_CMP_Cn_Start</td>
<td>Starts comparison for analog input voltage.</td>
</tr>
<tr>
<td>R_CMP_Cn_Stop</td>
<td>Ends comparison for analog input voltage.</td>
</tr>
</tbody>
</table>
**R_CMPC_Create**

Performs initialization necessary to control the Comparator C.

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

**[Syntax]**

```c
void R_CMPC_Create ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_CMPC_Create_UserInit

Performs user-defined initialization relating to the Comparator C.

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

[Syntax]

void R_CMPC_Create_UserInit ( void );

[Argument(s)]
None.

[Return value]
None.
r_cmpc_cmpcn_interrupt

Performs processing in response to the comparator C interrupt.

Remark This API function is called to run interrupt processing for the comparator \(C_n\) interrupt, which is generated when the comparison result changes at this time.

[Syntax]

\[
\text{static void } r\text{-cmpc}_n\text{-interrupt ( void );}
\]

Remark \(n\) is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_CMPCn_Start

Starts comparison for analog input voltage.

[Syntax]

```c
void R_CMPCn_Start ( void );
```

Remark  \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_CMPCn_Stop

Ends comparison for analog input voltage.

[Syntax]

```c
void R_CMPCn_Stop ( void );
```

Remark  

$n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
Usage example

Set the flag to notify changing the comparison result.

[GUI setting example]

<table>
<thead>
<tr>
<th>Comparator C</th>
<th>CMPC</th>
<th>Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparator C0</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>Comparator C1</td>
<td>Unused</td>
<td></td>
</tr>
<tr>
<td>Comparator C2</td>
<td>Unused</td>
<td></td>
</tr>
<tr>
<td>Comparator C3</td>
<td>Unused</td>
<td></td>
</tr>
<tr>
<td>Comparator C0 input select</td>
<td>AN000</td>
<td></td>
</tr>
<tr>
<td>Comparator C0 reference voltage setting</td>
<td>CVREFC0 (P20)</td>
<td></td>
</tr>
<tr>
<td>Comparator C0 enable digital filter</td>
<td>Unused</td>
<td></td>
</tr>
<tr>
<td>Comparator C0 output polarity</td>
<td>Normal</td>
<td></td>
</tr>
<tr>
<td>Comparator C0 enable output (COMP0)</td>
<td>Unused</td>
<td></td>
</tr>
<tr>
<td>Comparator C0 enable comparator C0 interrupt (CMPC0)</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>Comparator C0 edge select</td>
<td>Rising</td>
<td></td>
</tr>
<tr>
<td>Comparator C0 priority</td>
<td>Level 15 (highest)</td>
<td></td>
</tr>
</tbody>
</table>
[API setting example]

r_cg_main.c

void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start comparator C0 */
    R_CMPC0_Start();

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

r_cg_cmpc_user.c

/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_cmpc0_f;
/* End user code. Do not edit comment generated here */

void R_CMPC_Create_UserInit(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Clear the flag */
    g_cmpc0_f = 0U;
    /* End user code. Do not edit comment generated here */
}

static void r_cmpc_cmpc0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Set the flag */
    g_cmpc0_f = 1U;
    /* End user code. Do not edit comment generated here */
}
### 3.2.37 LCD controller / driver (LCD)

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>R_LCD_Create</code></td>
<td>Performs initialization necessary to control the LCD controller / driver.</td>
</tr>
<tr>
<td><code>R_LCD_Create_UserInit</code></td>
<td>Performs user-defined initialization relating to the LCD controller / driver.</td>
</tr>
<tr>
<td><code>R_LCD_Start</code></td>
<td>Sets the LCD controller / driver to display on status.</td>
</tr>
<tr>
<td><code>R_LCD_Stop</code></td>
<td>Sets the LCD controller / driver to display off status.</td>
</tr>
<tr>
<td><code>R_LCD_Voltage_On</code></td>
<td>Enables operation of internal voltage boost circuit and capacitor split circuit.</td>
</tr>
<tr>
<td><code>R_LCD_Voltage_Off</code></td>
<td>Disables operation of internal voltage boost circuit and capacitor split circuit.</td>
</tr>
</tbody>
</table>
**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]**

```c
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]**

```c
void R_LCD_Create_UserInit ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_LCD_Start

Sets the LCD controller / driver to display on status.

[Syntax]

void R_LCD_Start ( void );

[Argument(s)]
None.

[Return value]
None.
R_LCD_Stop

Sets the LCD controller / driver to display off status.

[Syntax]

```c
void R_LCD_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.
**R_LCD_Voltage_On**

Enables operation of internal voltage boost circuit and capacitor split circuit.

**[Syntax]**

```
void R_LCD_Voltage_On ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_LCD_Voltage_Off

Disables operation of internal voltage boost circuit and capacitor split circuit.

[Syntax]
void R_LCD_Voltage_Off ( void );

[Argument(s)]
None.

[Return value]
None.
### Revision Record

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>July 01, 2018</td>
<td>First Edition issued</td>
</tr>
</tbody>
</table>
Code Generator Tool