AP4
Integrated Development Environment
User's Manual: RZ/T1 API Reference

Target Device
RZ Family
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
Data 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 or registered trademarks in this document are the property of their respective owners.
# TABLE OF CONTENTS

1. **GENERAL** ................................................................. 5
   1.1 Overview .............................................................. 5
   1.2 Features .............................................................. 5

2. **OUTPUT FILES** ...................................................... 6
   2.1 Description .......................................................... 6

3. **API FUNCTIONS** .................................................... 13
   3.1 Overview ............................................................. 13
   3.2 Function Reference ............................................... 13
       3.2.1 Common ......................................................... 15
       3.2.2 Clock generator ............................................... 25
       3.2.3 Interrupt Controller ......................................... 30
       3.2.4 Bus State Controller ......................................... 40
       3.2.5 DMA Controller ................................................ 50
       3.2.6 Event Link Controller ......................................... 66
       3.2.7 I/O Ports ....................................................... 76
       3.2.8 Multi-Function Timer Pulse Unit .............................. 80
       3.2.9 Port Output Enable 3 ........................................... 102
       3.2.10 General PWM Timer ............................................ 112
       3.2.11 16-Bit Timer Pulse Unit ...................................... 130
       3.2.12 Programmable Pulse Generator ............................... 142
       3.2.13 Compare Match Timer .......................................... 146
       3.2.14 Compare Match Timer W ....................................... 152
       3.2.15 Watchdog Timer ................................................ 160
       3.2.16 Independent Watchdog Timer ................................ 166
       3.2.17 Serial Communications Interface with FIFO .................. 172
       3.2.18 \( \text{\textregistered} \text{C} \) Bus Interface ......................... 188
       3.2.19 Serial Peripheral Interface .................................. 208
       3.2.20 SPI Multi I/O Bus Controller ................................. 222
       3.2.21 CRC Operation Units .......................................... 230
       3.2.22 \( \Delta \Sigma \) Interface ........................................... 238
       3.2.23 Memory Protection Unit ....................................... 251
       3.2.24 Error Control Module ......................................... 254
       3.2.25 12-Bit A/D Converter .......................................... 306
       3.2.26 Data Operation Circuit ........................................ 318

Revision Record .......................................................... 326
1. GENERAL

The Code Generator is a software tool that automatically generates device drivers. This chapter gives an overview of the design tool.

1.1 Overview

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

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

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

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

This appendix describes the files output by the Code Generator.

2.1 Description

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

Table 2.1 Output File List

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>File Name</th>
<th>API Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common</td>
<td>r_cg_main.c</td>
<td>main</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_MAIN_UserInit</td>
</tr>
<tr>
<td></td>
<td>r_cg_mpc.c</td>
<td>R_MPC_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_MPC_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td>r_cg_systeminit.c</td>
<td>R_Systeminit</td>
</tr>
<tr>
<td></td>
<td>r_cg_intprg.c</td>
<td>r_set_exception_handler</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_fiq_handler</td>
</tr>
<tr>
<td></td>
<td>r_cg_macrosdriver.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_userdefine.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_interrupthandlers.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_mpc.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_nestintr_wrap.asm</td>
<td></td>
</tr>
<tr>
<td>Clock generator</td>
<td>r_cg_cgc.c</td>
<td>R_CGC_Create</td>
</tr>
<tr>
<td></td>
<td>r_cg_cgc_user.c</td>
<td>R_CGC_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td>r_cg_cgc.h</td>
<td></td>
</tr>
<tr>
<td>Interrupt Controller</td>
<td>r_cg_icu.c</td>
<td>R_ICU_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ICU_IRQn_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ICU_IRQn_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ICU_ETHPHYIn_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ICU_ETHPHYIn_Stop</td>
</tr>
<tr>
<td></td>
<td>r_cg_icu_user.c</td>
<td>R_ICU_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_icu_nmi_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_icu_irqn_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_icu_ethphyin_interrupt</td>
</tr>
<tr>
<td></td>
<td>r_cg_icu.h</td>
<td></td>
</tr>
<tr>
<td>Bus State Controller</td>
<td>r_cg_bsc.c</td>
<td>R_BSC_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_BSC.InitializeSDRAM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_BSC_SDRAMPowerDown_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_BSC_SDRAMPowerDown_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_BSC_SDRAMDeepPowerDown_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_BSC_SDRAMDeepPowerDown_Stop</td>
</tr>
<tr>
<td></td>
<td>r_cg_bsc_user.c</td>
<td>R_BSC_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_bsc_bscmci_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_bsc_tosf_interrupt</td>
</tr>
<tr>
<td></td>
<td>r_cg_bsc.h</td>
<td></td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>File Name</td>
<td>API Function Name</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| DMA Controller                    | r_cg_dmacc        | R_DMACn_Create
R_DMACn_Set_SoftwareTrigger
R_DMACm_Cn_Start
R_DMACm_Cn_Stop
R_DMACm_Cn_Suspend
R_DMACm_Cn_SuspendClear           |
|                                   | r_cg_dmacc_user.c | R_DMACn_Create_UserInit
r_dmacc0_dmaintn_interrupt
r_dmacc1_dmaint2_interrupt
r_dmacc0_dmarsrq0_interrupt
r_dmacc1_dmarsrq1_interrupt
r_dmacc0_dmaerr0_interrupt
r_dmacc1_dmaerr1_interrupt
r_callback_dmacc0_dmaintn_interrupt
r_callback_dmacc1_dmaint2_interrupt |
|                                   | r_cg_dmacc.h      | —                                                                                 |
| Event Link Controller             | r_cg_elc.c        | R_ELC_Create
R_ELC_Start
R_ELC_Stop
R_ELC_GenerateSoftwareEvent
R_ELC_Get_PortBuffern
R_ELC_Get_PortBuffern             |
|                                   | r_cg_elc_user.c   | R_ELC_Create_UserInit
r_elc_elcirqn_interrupt           |
|                                   | r_cg_elc.h        | —                                                                                 |
| I/O Ports                         | r_cg_port.c       | R_PORT_Create                                                                     |
|                                   | r_cg_port_user.c  | R_PORT_Create_UserInit                                                            |
|                                   | r_cg_port.h       | —                                                                                 |
| Multi-Function Timer Pulse Unit   | r_cg_mtu3.c       | R_MTU3_Create
R_MTU3_Cm_Start
R_MTU3_Cm_Stop                     |
|                                   | r_cg_mtu3_user.c  | R_MTU3_Create_UserInit
r_mtu3_tgiam_interrupt
r_mtu3_tgibm_interrupt
r_mtu3_tgicm_interrupt
r_mtu3_tgidm_interrupt
r_mtu3_tgie0_interrupt
r_mtu3_tgif0_interrupt
r_mtu3_tcivm_interrupt
r_mtu3_tcium_interrupt
r_mtu3_tgiu5_interrupt
r_mtu3_tgiv5_interrupt
r_mtu3_tgiw5_interrupt
r_mtu3_c4_tgia4_interrupt
r_mtu3_c4_tgib4_interrupt
r_mtu3_c4_tciv4_interrupt
r_mtu3_c7_tgia7_interrupt
r_mtu3_c7_tgib7_interrupt
r_mtu3_c7_tciv7_interrupt          |
<p>|                                   | r_cg_mtu3.h       | —                                                                                 |</p>
<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>File Name</th>
<th>API Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port Output Enable 3</td>
<td>r_cg_poe3.c</td>
<td>R_POE3_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_POE3_Start</td>
</tr>
<tr>
<td></td>
<td>r_cg_poe3_user.c</td>
<td>R_POE3_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td>r_cg_poe3.h</td>
<td></td>
</tr>
<tr>
<td>General PWM Timer</td>
<td>r_cg_gpt.c</td>
<td>R_GPT_Create</td>
</tr>
<tr>
<td></td>
<td>r_cg_gpt_user.c</td>
<td>R_GPT_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td>r_cg_gpt.h</td>
<td></td>
</tr>
<tr>
<td>16-Bit Timer Pulse Unit</td>
<td>r_cg_tpu.c</td>
<td>R_TPU_Create</td>
</tr>
<tr>
<td></td>
<td>r_cg_tpu_user.c</td>
<td>R_TPU_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td>r_cg_tpu.h</td>
<td></td>
</tr>
<tr>
<td>Programmable Pulse Generator</td>
<td>r_cg_ppg.c</td>
<td>R_PPG_Create</td>
</tr>
<tr>
<td></td>
<td>r_cg_ppg_user.c</td>
<td>R_PPG_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td>r_cg_ppg.h</td>
<td></td>
</tr>
<tr>
<td>Compare Match Timer</td>
<td>r_cg_cmt.c</td>
<td>R_CMTn_Create</td>
</tr>
<tr>
<td></td>
<td>r_cg_cmt_user.c</td>
<td>R_CMTn_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td>r_cg_cmt.h</td>
<td></td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>File Name</td>
<td>API Function Name</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>----------------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td><strong>Compare Match Timer W</strong></td>
<td>r_cg_cmtw.c</td>
<td>R_CMTWm_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_CMTWm_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_CMTWm_Stop</td>
</tr>
<tr>
<td></td>
<td>r_cg_cmtw_user.c</td>
<td>R_CMTWm_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_cmtw_cmwim_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_cmtw_icnim_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_cmtw_ocnim_interrupt</td>
</tr>
<tr>
<td></td>
<td>r_cg_cmtw.h</td>
<td>--</td>
</tr>
<tr>
<td><strong>Watchdog Timer</strong></td>
<td>r_cg_wdt.c</td>
<td>R_WDTn_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_WDTn_Restart</td>
</tr>
<tr>
<td></td>
<td>r_cg_wdt_user.c</td>
<td>R_WDTn_CREATE_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_wdtn_undff_refef_interrupt</td>
</tr>
<tr>
<td></td>
<td>r_cg_wdt.h</td>
<td>--</td>
</tr>
<tr>
<td><strong>Independent Watchdog Timer</strong></td>
<td>r_cg_iwdt.c</td>
<td>R_IWDT_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_IWDT_Restart</td>
</tr>
<tr>
<td></td>
<td>r_cg_iwdt_user.c</td>
<td>R_IWDT_CREATE_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_iwdt_undff_refef_interrupt</td>
</tr>
<tr>
<td></td>
<td>r_cg_iwdt.h</td>
<td>--</td>
</tr>
<tr>
<td><strong>Serial Communications Interface with FIFO</strong></td>
<td>r_cg_scifa.c</td>
<td>R_SCIFAn_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_SCIFAn_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_SCIFAn_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_SCIFAn_Serial_Send</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_SCIFAn_Serial_Receive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_SCIFAn_Serial_Send_Receive</td>
</tr>
<tr>
<td></td>
<td>r_cg_scifa_user.c</td>
<td>R_SCIFAn_CREATE_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_scifan_txifn_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_scifan_rxifn_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_scifan_brifn_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_scifan_drifn_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_scifan_telfn_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_scifan_erifn_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_scifan_callback_transmitend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_scifan_callback_receiveend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_scifan_callback_error</td>
</tr>
<tr>
<td></td>
<td>r_cg_scifa.h</td>
<td>--</td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>File Name</td>
<td>API Function Name</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>I^2^C Bus Interface</strong></td>
<td>r_cg_riic.c</td>
<td>R_RIICn_Create&lt;br&gt;R_RIICn_Start&lt;br&gt;R_RIICn_Stop&lt;br&gt;R_RIICn_Master_Send&lt;br&gt;R_RIICn_Master_Send_Without_Stop&lt;br&gt;R_RIICn_Master_Receive&lt;br&gt;R_RIICn_Slave_Send&lt;br&gt;R_RIICn_Slave_Receive&lt;br&gt;R_RIICn_StartCondition&lt;br&gt;R_RIICn_StopCondition</td>
</tr>
<tr>
<td></td>
<td>r_cg_riic_user.c</td>
<td>R_RIICn_Create_UserInit&lt;br&gt;r_riic_error_interrupt&lt;br&gt;r_riic_receive_interrupt&lt;br&gt;r_riic_transmit_interrupt&lt;br&gt;r_riic_transmitend_interrupt&lt;br&gt;r_riic_callback_receiveerror&lt;br&gt;r_riic_callback_transmitend&lt;br&gt;r_riic_callback_receiveend</td>
</tr>
<tr>
<td></td>
<td>r_cg_riic.h</td>
<td>–</td>
</tr>
<tr>
<td><strong>Serial Peripheral Interface</strong></td>
<td>r_cg_rspi.c</td>
<td>R_RSPIn_Create&lt;br&gt;R_RSPIn_Start&lt;br&gt;R_RSPIn_Stop&lt;br&gt;R_RSPIn_Send&lt;br&gt;R_RSPIn_Send_Receive</td>
</tr>
<tr>
<td></td>
<td>r_cg_rspi_user.c</td>
<td>R_RSPIn_Create_UserInit&lt;br&gt;r_rspin_receive_interrupt&lt;br&gt;r_rspin_transmit_interrupt&lt;br&gt;r_rspin_error_interrupt&lt;br&gt;r_rspin_idle_interrupt&lt;br&gt;r_rspin_callback_receiveend&lt;br&gt;r_rspin_callback_error&lt;br&gt;r_rspin_callback_transmitend</td>
</tr>
<tr>
<td></td>
<td>r_cg_rspi.h</td>
<td>–</td>
</tr>
<tr>
<td><strong>SPI Multi I/O Bus Controller</strong></td>
<td>r_cg_spibsc.c</td>
<td>R_SPIBSC_Create&lt;br&gt;R_SPIBSC_EAVUpperAddressChange&lt;br&gt;R_SPIBSC_SPIRead&lt;br&gt;R_SPIBSC_SPIWrite&lt;br&gt;R_SPIBSC_SPIRead_Write</td>
</tr>
<tr>
<td></td>
<td>r_cg_spibsc_user.c</td>
<td>R_SPIBSC_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td>r_cg_spibsc.h</td>
<td>–</td>
</tr>
<tr>
<td><strong>CRC Operation Units</strong></td>
<td>r_cg_crc.c</td>
<td>R_CRC_SetCRC8_2F&lt;br&gt;R_CRC_SetCRC8_SAE&lt;br&gt;R_CRC_SetCRC16_CCITT&lt;br&gt;R_CRC_SetCRC32_ETHER&lt;br&gt;R_CRC_Input_Data&lt;br&gt;R_CRC_Get_Result</td>
</tr>
<tr>
<td></td>
<td>r_cg_crc.h</td>
<td>–</td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>File Name</td>
<td>API Function Name</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ΔΣ Interface</td>
<td>r_cg_dsmif.c</td>
<td>R_DSMIF_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DSMIF_UVW_Start</td>
</tr>
<tr>
<td></td>
<td>r_cg_dsmif_user.c</td>
<td>R_DSMIF_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td>r_cg_dsmif.h</td>
<td>--</td>
</tr>
<tr>
<td>Error Control Module</td>
<td>r_cg_ecm.c</td>
<td>R_ECM_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_WDT0_Error_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_WDT0_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_IWDTa_Error_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_IWDTa_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_CGC_Error_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_CGC_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit0_Error_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit0_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ECM_Pseudo_ADC_Unit1_Error_STOP</td>
</tr>
<tr>
<td></td>
<td>r_cg_ECM_user.c</td>
<td>R_ECM_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td>r_cg_ecm_user.c</td>
<td>r_ecm_nmi_interrupt</td>
</tr>
<tr>
<td></td>
<td>r_cg_ecm_user.c</td>
<td>r_ecm_errd_interrupt</td>
</tr>
<tr>
<td></td>
<td>r_cg_ecm_user.c</td>
<td>r_ecm_compareerror_interrupt</td>
</tr>
<tr>
<td></td>
<td>r_cg_ecm.h</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>r_cg_ecm.h</td>
<td>--</td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>File Name</td>
<td>API Function Name</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| 12-Bit A/D Converter     | `r_cg_s12ad.c`          | R_S12ADn_Create  
                        R_S12ADn_Start  
                        R_S12ADn_Stop   
                        R_S12ADn_Get_ValueResult  
                        R_S12ADn_Set_CompareValue |
|                          | `r_cg_s12ad_user.c`     | R_S12ADn_Create_UserInit  
                        r_s12ad_s12adn_interrupt  
                        r_s12ad_s12gbadin_interrupt  
                        r_s12ad_s12cmpn_interrupt  
                        r_s12ad_s12aden_interrupt |
|                          | `r_cg_s12ad.h`          | -                                                                               |
| Data Operation Circuit   | `r_cg_doc.c`            | R_DOC_Create  
                        R_DOC_SetMode  
                        R_DOC_WriteData  
                        R_DOC_GetResult  
                        R_DOC_ClearFlag |
|                          | `r_cg_doc_user.c`       | R_DOC_Create_UserInit  
                        r_doc_dopcf_interrupt |
|                          | `r_cg_doc.h`            | -                                                                               |
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.

3.2 Function Reference

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

Figure 3.1 Notation Format of API Functions

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

(2) Outline
Outlines the functions of the API function.

(3) [Syntax]
Indicates the format to be used when describing an API function to be called in C language.
(4) [Argument(s)]
API function arguments are explained in the following format.

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_MPC_Create</td>
<td>Performs initialization necessary to control the multi-function pin controller.</td>
</tr>
<tr>
<td>R_MPC_Create_UserInit</td>
<td>Performs user-defined initialization relating to the multi-function pin controller.</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_Systeminit</td>
<td>Performs initialization necessary to control the various peripheral functions.</td>
</tr>
<tr>
<td>r_set_exception_handler</td>
<td>Sets FIQ exception handler.</td>
</tr>
<tr>
<td>r_fiq_handler</td>
<td>Performs FIQ exception.</td>
</tr>
<tr>
<td>NESTED_INTERRUPT_PUSH</td>
<td>Push the register value in response to the nested interrupt.</td>
</tr>
<tr>
<td>NESTED_INTERRUPT_POP</td>
<td>Pop the register value in response to the nested interrupt.</td>
</tr>
</tbody>
</table>
**R_MPC_Create**

Performs initialization necessary to control the multi-function pin controller.

**[Syntax]**

```c
void R_MPC_Create ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_MPC_Create_UserInit

Performs user-defined initialization relating to the multi-function pin controller.

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

[Syntax]

```c
void     R_MPC_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.
This is a main function.

Remark Call this API function from the startup routine.

[Syntax]

```c
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]

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

[Argument(s)]

None.

[Return value]

None.
R_Systeminit

Performs initialization necessary to control the various peripheral functions.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
Sets exception handler.

**Syntax**

```c
void r_set_exception_handler ( void );
```

**Argument(s)**

None.

**Return value**

None.
r_fiq_handler

Performs FIQ handler.

[Syntax]

```c
void r_fiq_handler ( void );
```

[Argument(s)]

None.

[Return value]

None.
NESTED_INTERRUPT_PUSH

Push the register value in response to the nested interrupt.

[Syntax]

```c
void NESTED_INTERRUPT_PUSH ( void );
```

[Argument(s)]

None.

[Return value]

None.
NESTED_INTERRUPT_POP

Pop the register value in response to the nested interrupt.

**[Syntax]**

```c
void NESTED_INTERRUPT_POP ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
### 3.2.2 Clock generator

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

<table>
<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 generator (include reset function, on-chip debug function, etc.).</td>
</tr>
<tr>
<td>R_CGC_Create_UserInit</td>
<td>Performs user-defined initialization relating to the clock generator (include reset function, on-chip debug function, etc.).</td>
</tr>
<tr>
<td>r_cgc_ostde_interrupt</td>
<td>Performs processing in response to the main clock oscillation stop detection interrupt.</td>
</tr>
</tbody>
</table>
R_CGC_Create

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

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

**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_ostde_interrupt

Performs processing in response to the main clock oscillation stop detection interrupt.

[Syntax]

```c
void r_cgc_ostde_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.
3.2.3 Interrupt Controller

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 required 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_Start</td>
<td>Enables IRQ&lt;sub&gt;n&lt;/sub&gt; interrupt.</td>
</tr>
<tr>
<td>R_ICU_IRQn_Stop</td>
<td>Disables IRQ&lt;sub&gt;n&lt;/sub&gt; interrupt.</td>
</tr>
<tr>
<td>R_ICU_ETHPHYIn_Start</td>
<td>Enables Ether PHY interrupt (EHTPHY&lt;sub&gt;n&lt;/sub&gt;).</td>
</tr>
<tr>
<td>R_ICU_ETHPHYIn_Stop</td>
<td>Disables Ether PHY interrupt (EHTPHY&lt;sub&gt;n&lt;/sub&gt;).</td>
</tr>
<tr>
<td>r_icu_nmi_interrupt</td>
<td>Clears NMI non-maskable interrupt request.</td>
</tr>
<tr>
<td>r_icu_irqn_interrupt</td>
<td>Performs processing in response to the IRQ&lt;sub&gt;n&lt;/sub&gt; interrupt.</td>
</tr>
<tr>
<td>r_icu_ethphyin_interrupt</td>
<td>Performs processing in response to the EHTPHY&lt;sub&gt;n&lt;/sub&gt;.</td>
</tr>
</tbody>
</table>
R_ICU_Create

Performs initialization required to control the interrupt controller.

[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]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_ICU_IRQn_Start

Enables IRQn interrupt.

[Syntax]

void R_ICU_IRQn_Start ( void );

Remark $n$ is the interrupt source number.

[Argument(s)]

None.

[Return value]

None.
# R_ICU_IRQn_Stop

Disables IRQn interrupt.

**[Syntax]**

```c
void R_IRQn_Stop ( void );
```

**Remark**

$n$ is the interrupt source number.

**[Argument(s)]**

None.

**[Return value]**

None.
Enables Ether PHY interrupt (EHTPHY\textsubscript{n}).

**[Syntax]**

```c
void R_ICU_ETHPHYIn_Start ( void );
```

Remark \( n \) is the interrupt source number.

**[Argument(s)]**

None.

**[Return value]**

None.
Disables Ether PHY interrupt (EHTPHY$n$).

**[Syntax]**

```c
void R_ICU_ETHPHYIn_Stop ( void );
```

Remark  
$n$ is the interrupt source number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_icu_nmi_interrupt

Clears NMI non-maskable interrupt request.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
r_icu_irqn_interrupt

Performs processing in response to the IRQn interrupt.

**[Syntax]**

```c
void r_icu_irqn_interrupt ( void );
```

Remark \( n \) is the interrupt source number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_icu_ethphyin_interrupt

Performs processing in response to the EHTPHYIn.

[Syntax]

```c
void r_icu_ethphyin_interrupt ( void );
```

Remark

\( n \) is the interrupt source number.

[Argument(s)]

None.

[Return value]

None.
### 3.2.4 Bus State Controller

Below is a list of API functions output by the Code Generator for Bus State Controller use.

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_BSC_Create</td>
<td>Performs initialization required to control the bus state controller.</td>
</tr>
<tr>
<td>R_BSC_Create_UserInit</td>
<td>Performs user-defined initialization relating to the bus state controller.</td>
</tr>
<tr>
<td>r_bsc_bsccmi_interrupt</td>
<td>Performs processing in response to the compare match interrupt.</td>
</tr>
<tr>
<td>r_bsc_tosf_interrupt</td>
<td>Performs processing in response to the error of detecting a long access wait state caused by the external WAIT pin.</td>
</tr>
<tr>
<td>R_BSC_InitializeSDRAM</td>
<td>Performs initialization required to control the SDRAM.</td>
</tr>
<tr>
<td>R_BSC_SDRAMPowerDown_Start</td>
<td>After the SDRAM is accessed, the SDRAM enters the power-down mode.</td>
</tr>
<tr>
<td>R_BSC_SDRAMPowerDown_Stop</td>
<td>After the SDRAM is accessed, the SDRAM does not enters the power-down mode.</td>
</tr>
<tr>
<td>R_BSC_SDRAMDeepPowerDown_Start</td>
<td>After the SDRAM is accessed, the SDRAM enters the deep power-down mode.</td>
</tr>
<tr>
<td>R_BSC_SDRAMDeepPowerDown_Stop</td>
<td>The low-power SDRAM enters the self-refresh mode.</td>
</tr>
</tbody>
</table>
### R_BSC_Create

Performs initialization required to control the bus state controller.

#### Syntax

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

#### Argument(s)

None.

#### Return value

None.
**R_BSC_Create_UserInit**

Performs user-defined initialization relating to the bus state controller.

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

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_bsc_bscmcmi_interrupt

Performs processing in response to the compare match interrupt.

[Syntax]

```
void r_bsc_bscmcmi_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.
r_bsc_tostf_interrupt

Performs processing in response to the error of detecting a long access wait state caused by the external WAIT pin.

Syntax

```c
void r_bsc_tostf_interrupt ( void );
```

Argument(s)

None.

Return value

None.
### R_BSC_InitializeSDRAM

Performs initialization required to control the SDRAM.

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
After the SDRAM is accessed, the SDRAM enters the power-down mode.

[R_BSC_SDRAMPowerDown_Start]

Syntax

```c
void R_BSC_SDRAMPowerDown_Start ( void );
```

Argument(s)

None.

Return value

None.
After the SDRAM is accessed, the SDRAM does not enter the power-down mode.

[Syntax]

```c
void R_BSC_SDRAMPowerDown_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.
After the SDRAM is accessed, the SDRAM enters the deep power-down mode.

**[Syntax]**

```c
void R_BSC_SDRAMDeepPowerDown_Start ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
3. GENERAL

R_BSC_SDRAMDeepPowerDown_Stop

The low-power SDRAM enters the self-refresh mode.

[Syntax]

void R_BSC_SDRAMDeepPowerDown_Start ( void );

[Argument(s)]

None.

[Return value]

None.
3.2.5 DMA Controller

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

Table 3.6 API Functions: [DMA Controller]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_DMACn_Create</td>
<td>Performs initialization required to control the DMA controller.</td>
</tr>
<tr>
<td>R_DMACn_Create_UserInit</td>
<td>Performs user-defined initialization relating to the DMA controller.</td>
</tr>
<tr>
<td>r_dmac0_dmaintn_interrupt</td>
<td>Performs processing in response to the external DMA request interrupt n of DMAC0.</td>
</tr>
<tr>
<td>r_dmac1_dmaint2_interrupt</td>
<td>Performs processing in response to the external DMA request interrupt 2 of DMAC1.</td>
</tr>
<tr>
<td>r_dmac0_dmasrq0_interrupt</td>
<td>Performs processing in response to the external DMA software activation request 0 interrupt of DMAC0.</td>
</tr>
<tr>
<td>r_dmac1_dmasrq1_interrupt</td>
<td>Performs processing in response to the external DMA software activation request 1 interrupt of DMAC1.</td>
</tr>
<tr>
<td>r_dmac0_dmaerr0_interrupt</td>
<td>Performs processing in response to the external DMA transfer error 0 interrupt of DMAC0.</td>
</tr>
<tr>
<td>r_dmac1_dmaerr1_interrupt</td>
<td>Performs processing in response to the external DMA transfer error 1 interrupt of DMAC1.</td>
</tr>
<tr>
<td>r_callback_dmac0_dmaintn_interrupt</td>
<td>Performs call-back processing in response to the external DMA request n interrupt (n=0~1) of DMAC0.</td>
</tr>
<tr>
<td>r_callback_dmac1_dmaint2_interrupt</td>
<td>Performs call-back processing in response to the external DMA request 2 interrupt of DMAC1.</td>
</tr>
<tr>
<td>R_DMACn_Set_SoftwareTrigger</td>
<td>Requests the DMA transfer.</td>
</tr>
<tr>
<td>R_DMAMc_Cn_Start</td>
<td>Enables the DMA transfer of DMA Channel n.</td>
</tr>
<tr>
<td>R_DMAMc_Cn_Stop</td>
<td>Halts the DMA transfer of DMA Channel n.</td>
</tr>
<tr>
<td>R_DMAMc_Cn_Suspend</td>
<td>Suspends the DMA transfer of DMA Channel n.</td>
</tr>
<tr>
<td>R_DMAMc_Cn_SuspendClear</td>
<td>Clears the temporary suspend status of the DMA transfer.</td>
</tr>
</tbody>
</table>
**R_DMACn_Create**

Performs initialization required to control the DMA controller.

**[Syntax]**

```c
void R_DMACn_Create ( void );
```

Remark: \( n \) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_DMACn_Create_UserInit**

Performs user-defined initialization relating to the DMA controller.

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

**[Syntax]**

```c
void R_DMACn_Create_UserInit ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_dmac0_dmaintn_interrupt

Performs processing in response to the external DMA request \( n \) interrupt of DMAC0.

[Syntax]

```c
void r_dmac0_dmaintn_interrupt ( void );
```

Remark \( n \) is the request signal number.

[Argument(s)]

None.

[Return value]

None.
r_dmac1_dmaint2_interrupt

Performs processing in response to the external DMA request 2 interrupt of DMAC1.

[Syntax]

```c
void r_dmac1_dmaint2_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.
r_dmac0_dmasrq0_interrupt

Performs processing in response to the external DMA software activation request 0 interrupt of DMAC0.

[Syntax]

```c
void r_dmac0_dmasrq0_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.
r_dmac1_dmasrq1_interrupt

Performs processing in response to the external DMA software activation request 1 interrupt of DMAC1.

[Syntax]

```c
void r_dmac1_dmasrq1_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.
r_dmac0_dmaerr0_interrupt

Performs processing in response to the external DMA transfer error 0 interrupt of DMAC0.

[Syntax]

```c
void r_dmac0_dmaerr0_interrupt ( void );
```

[Argument(s)]
None.

[Return value]
None.
**r_dmac1_dmaerr1_interrupt**

Performs processing in response to the external DMA transfer error 1 interrupt of DMAC1.

**[Syntax]**

```c
void r_dmac1_dmaerr1_interrupt ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
**r_callback_dmac0_dmaintn_interrupt**

Performs call-back processing in response to the external DMA request \( n \) interrupt (\( n=0\sim1 \)) of DMAC0.

**[Syntax]**

```c
void r_callback_dmac0_dmaintn_interrupt ( void );
```

Remark \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
Performs call-back processing in response to the external DMA request 2 interrupt of DMAC1.

**[Syntax]**

```c
void r_callback_dmac1_dmaint2_interrupt ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_DMACn_Set_SoftwareTrigger

Requests the DMA transfer.

[Syntax]

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

Remark  

$m$ is the unit number.

[Argument(s)]

None.

[Return value]

None.
R_DMACm_Cn_Start

Enables the DMA transfer of DMA Channel n.

[Syntax]

```c
void R_DMACm_Cn_Start ( void );
```

Remark  
\( m \) is the unit number, \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
Halts the DMA transfer of DMA Channel n.

**Syntax**

```c
void R_DMACm_Cn_Stop ( void );
```

Remark  

$m$ is the unit number, $n$ is the channel number.

**Argument(s)**

None.

**Return value**

None.
### R_DMACm_Cn_Suspend

Suspends the DMA transfer of DMA Channel n.

**[Syntax]**

```c
void R_DMACm_Cn_Suspend ( void );
```

**Remark**

$m$ is the unit number, $n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_DMACm_Cn_SuspendClear

Clears the temporary suspend status of the DMA transfer.

[Syntax]

```c
void R_DMACm_Cn_SuspendClear ( void );
```

Remark

\( m \) is the unit number, \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
### 3.2.6 Event Link Controller

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 required 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_elcirqn_interrupt</td>
<td>Operation disable of the event link controller.</td>
</tr>
<tr>
<td>R_ELC_Start</td>
<td>Enables the ELC function.</td>
</tr>
<tr>
<td>R_ELC_Stop</td>
<td>Disables the ELC function.</td>
</tr>
<tr>
<td>R_ELC_GenerateSoftwareEvent</td>
<td>Disabled the software event.</td>
</tr>
<tr>
<td>R_ELC_Get_PortBuffern</td>
<td>Reads the value to the port buffer register (PDBFn).</td>
</tr>
<tr>
<td>R_ELC_Set_PortBuffern</td>
<td>Writes the value to the port buffer register (PDBFn).</td>
</tr>
</tbody>
</table>
R_ELC_Create

Performs initialization required to control the event link controller.

[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_elcirqn_interrupt

Operation disable of the event link controller.

[Syntax]

```c
void r_elc_elcirqn_interrupt ( void );
```

Remark $n$ is the port group number.

[Argument(s)]

None.

[Return value]

None.
R_ELC_Start

Enables the ELC function.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
Disables the ELC function.

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_ELC_GenerateSoftwareEvent

Disabled the software event.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
R_ELC_Get_PortBuffer\text{n}

Reads the value to the port buffer register (PDBF\text{n}).

[Syntax]

```c
void R_ELC_Get_PortBuffer\text{n} ( uint8_t * const value );
```

[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 value;</td>
<td>The pointer in which the value of the buffer register is stored.</td>
</tr>
</tbody>
</table>

Remark \(n\) is the port group number.

[Return value]

None.
**R_ELC_Set_PortBuffer n**

Writes the value to the port buffer register (PDBF n).

**Syntax**

```c
void R_ELC_Set_PortBuffer n ( uint8_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>uint8_t</td>
<td>value; The write value in the buffer register</td>
</tr>
</tbody>
</table>

**Remark**  

n is the port group number.

**Return value**

None.
3.2.7 I/O Ports

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

Table 3.8 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 required to control the I/O port.</td>
</tr>
<tr>
<td>R_PORT_Create_UserInit</td>
<td>Performs user-defined initialization relating to the I/O port.</td>
</tr>
</tbody>
</table>
R_PORT_Create

Performs initialization required to control the I/O port.

[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 port.

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
### 3.2.8 Multi-Function Timer Pulse Unit

Below is a list of API functions output by the Code Generator for Multi-Function Timer Pulse Unit use.

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>R_MTU3_Create</code></td>
<td>Performs initialization required to control the multi-function timer pulse unit.</td>
</tr>
<tr>
<td><code>R_MTU3_Create_UserInit</code></td>
<td>Performs user-defined initialization relating to the multi-function timer pulse unit.</td>
</tr>
<tr>
<td><code>r_mtu3_tgi4m_interrupt</code></td>
<td>Performs processing in response to the channel$m$ input capture/compare match A interrupt.</td>
</tr>
<tr>
<td><code>r_mtu3_tgibm_interrupt</code></td>
<td>Performs processing in response to the channel$m$ input capture/compare match B interrupt.</td>
</tr>
<tr>
<td><code>r_mtu3_tgicm_interrupt</code></td>
<td>Performs processing in response to the channel$m$ input capture/compare match C interrupt.</td>
</tr>
<tr>
<td><code>r_mtu3_tgidm_interrupt</code></td>
<td>Performs processing in response to the channel$m$ input capture/compare match D interrupt.</td>
</tr>
<tr>
<td><code>r_mtu3_tgie0_interrupt</code></td>
<td>Performs processing in response to the channel0 input capture/compare match E interrupt.</td>
</tr>
<tr>
<td><code>r_mtu3_tgif0_interrupt</code></td>
<td>Performs processing in response to the channel0 input capture/compare match F interrupt.</td>
</tr>
<tr>
<td><code>r_mtu3_tcivm_interrupt</code></td>
<td>Performs processing in response to the channel$m$ overflow interrupt.</td>
</tr>
<tr>
<td><code>r_mtu3_tcium_interrupt</code></td>
<td>Performs processing in response to the channel$m$ underflow interrupt.</td>
</tr>
<tr>
<td><code>r_mtu3_tgiu5_interrupt</code></td>
<td>Performs processing in response to the channel5 input capture/compare match U interrupt.</td>
</tr>
<tr>
<td><code>r_mtu3_tgiv5_interrupt</code></td>
<td>Performs processing in response to the channel5 input capture/compare match V interrupt.</td>
</tr>
<tr>
<td><code>r_mtu3_tgiw5_interrupt</code></td>
<td>Performs processing in response to the channel5 input capture/compare match W interrupt.</td>
</tr>
<tr>
<td><code>r_mtu3_c4_tgia4_interrupt</code></td>
<td>Performs processing in response to the TGIA4 interrupt.</td>
</tr>
<tr>
<td><code>r_mtu3_c4_tgib4_interrupt</code></td>
<td>Performs processing in response to the TGiB4 interrupt.</td>
</tr>
<tr>
<td><code>r_mtu3_c4_tciv4_interrupt</code></td>
<td>Performs processing in response to the TCIV4 interrupt.</td>
</tr>
<tr>
<td><code>r_mtu3_c7_tgia7_interrupt</code></td>
<td>Performs processing in response to the TGIA7 interrupt.</td>
</tr>
<tr>
<td><code>r_mtu3_c7_tgib7_interrupt</code></td>
<td>Performs processing in response to the TGiB7 interrupt.</td>
</tr>
<tr>
<td><code>r_mtu3_c7_tciv7_interrupt</code></td>
<td>Performs processing in response to the TCIV7 interrupt.</td>
</tr>
<tr>
<td><code>R_MTU3_Cm_Start</code></td>
<td>Starts the count of the channel $m$.</td>
</tr>
<tr>
<td><code>R_MTU3_Cm_Stop</code></td>
<td>Ends the count of the channel $m$.</td>
</tr>
</tbody>
</table>
R_MTU3_Create

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

[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.

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

**[Syntax]**

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

**[Argument(s)]**

- None.

**[Return value]**

- None.
### r_mtu3_tgiam_interrupt

Performs processing in response to the channel\(m\) input capture/compare match A interrupt.

**Remark** This API function is called as the interrupt process corresponding to the TGIA\(m\) interrupt.

**[Syntax]**

```c
void r_mtu3_tgiam_interrupt ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_mtu3_tgibm_interrupt

Performs processing in response to the channel\textit{m} input capture/compare match B interrupt.

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

**[Syntax]**

```c
void r_mtu3_tgibm_interrupt ( void );
```

Remark \textit{m} is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
Performs processing in response to the channel $m$ input capture/compare match C interrupt.

Remark   This API function is called as the interrupt process corresponding to the TGIC$m$ interrupt.

**[Syntax]**

```c
void   r_mtu3_tgicm_interrupt ( void );
```

Remark   $m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_mtu3_tgidm_interrupt

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

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

Syntax

```c
void r_mtu3_tgidm_interrupt ( void );
```

Remark m is the channel number.

Argument(s)

None.

Return value

None.
**r_mtu3_tgie0_interrupt**

Performs processing in response to the channel0 input capture/compare match E interrupt.

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

**[Syntax]**

```c
void r_mtu3_tgie0_interrupt ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
r_mtu3_tgif0_interrupt

Performs processing in response to the channel0 input capture/compare match F interrupt.

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

[Syntax]

```c
void r_mtu3_tgif0_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.
### r_mtu3_tcivm_interrupt

Performs processing in response to the channel $m$ overflow interrupt.

**Remark**  This API function is called as the interrupt process corresponding to the TCIV$m$ interrupt.

**[Syntax]**

```c
void r_mtu3_tcivm_interrupt ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
Perform processing in response to the channel $m$ underflow interrupt.

Remark This API function is called as the interrupt process corresponding to the TCIU$m$ interrupt.

**[Syntax]**

```c
void r_mtu3_tciu_m_interrupt ( void );
```

Remark $m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_mtu3_tgiu5_interrupt

Performs processing in response to the channel5 input capture/compare match U interrupt.

Remark  This API function is called as the interrupt process corresponding to the TGIU5 interrupt.

[Syntax]

```c
void r_mtu3_tgiu5_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.
r_mtu3_tgiv5_interrupt

Performs processing in response to the channel5 input capture/compare match V interrupt.

Remark This API function is called as the interrupt process corresponding to the TGIV5 interrupt.

[Syntax]

```c
void r_mtu3_tgiv5_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.
r_mtu3_tgiw5_interrupt

Performs processing in response to the channel5 input capture/compare match W interrupt.

Remark This API function is called as the interrupt process corresponding to the TGIW5 interrupt.

[Syntax]

```c
void r_mtu3_tgiw5_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.
r_mtu3_c4_tgia4_interrupt

Performs processing in response to the TGIA4 interrupt.

Remark This API function is called as the interrupt process corresponding to the TGIA4 interrupt.

[Syntax]

```c
void r_mtu3_c4_tgia4_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.
r_mtu3_c4_tgib4_interrupt

Performs processing in response to the TGIB4 interrupt.

Remark This API function is called as the interrupt process corresponding to the TGIB4 interrupt.

[Syntax]

void    r_mtu3_c4_tgib4_interrupt ( void );

[Argument(s)]

None.

[Return value]

None.
**r_mtu3_c4_tciv4_interrupt**

Performs processing in response to the TCIV4 interrupt.

**Remark**  This API function is called as the interrupt process corresponding to the TCIV4 interrupt.

**[Syntax]**

```c
void r_mtu3_c4_tciv4_interrupt ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
r_mtu3_c7_tgia7_interrupt

Performs processing in response to the TGIA7 interrupt.

Remark This API function is called as the interrupt process corresponding to the TGIA7 interrupt.

[Syntax]

```c
void r_mtu3_c7_tgia7_interrupt ( void );
```

[Argument(s)]
None.

[Return value]
None.
**r_mtu3_c7_tgib7_interrupt**

Performs processing in response to the TGIB7 interrupt.

**Remark**  This API function is called as the interrupt process corresponding to the TGIB7 interrupt.

**[Syntax]**

```c
void r_mtu3_c7_tgib7_interrupt ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
r_mtu3_c7_tciv7_interrupt

Perform processing in response to the TCIV7 interrupt.

Remark This API function is called as the interrupt process corresponding to the TCIA7 interrupt.

Syntax

```c
void r_mtu3_c7_tciv7_interrupt ( void );
```

Argument(s)

None.

Return value

None.
**R_MTU3_Cm_Start**

Starts the count of the channel \( m \).

**[Syntax]**

```c
void R_MTU3_Cm_Start ( void );
```

Remark \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_MTU3_Cm_Stop

Ends the count of the channel \( m \).

[Syntax]

```c
void R_MTU3_Cm_Stop ( void );
```

Remark \( m \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
### 3.2.9 Port Output Enable 3

Below is a list of API functions output by the Code Generator for Port Output Enable use.

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_POE3_Create</td>
<td>Performs initialization required to control the port output enable3.</td>
</tr>
<tr>
<td>R_POE3_Create_UserInit</td>
<td>Performs user-defined initialization relating to the port output enable3.</td>
</tr>
<tr>
<td>r_poe3_oein_interrupt</td>
<td>Performs processing in response to the output enable interrupt.</td>
</tr>
<tr>
<td>R_POE3_Start</td>
<td>Places the pin in high-impedance state.</td>
</tr>
<tr>
<td>R_POE3_Stop</td>
<td>Disable the pin in high-impedance state.</td>
</tr>
<tr>
<td>R_POE3_Set_HiZ_MTUm</td>
<td>Places the MTUm pin in high-impedance state. Some value of m may cause MTU pins or GPT pins to be in high-impedance state.</td>
</tr>
<tr>
<td>R_POE3_Clear_HiZ_MTUm</td>
<td>Disables the MTUm pin in high-impedance state. Some value of m may cause MTU pins or GPT pins in high-impedance state to be disabled.</td>
</tr>
<tr>
<td>R_POE3_Set_HiZ_GPT3</td>
<td>Places the GPT3 pin in high-impedance state.</td>
</tr>
<tr>
<td>R_POE3_Clear_HiZ_GPT3</td>
<td>Disables the GPT3 pin in high-impedance state.</td>
</tr>
</tbody>
</table>
**R_POE3_Create**

Performs initialization required to control the port output enable3.

**[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 enable3.

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.
Performs processing in response to the output enable interrupt.

**[Syntax]**

```c
void r_poe3_oein_interrupt ( void );
```

Remark $n$ is the interrupt source number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_POE3_Start**

Places the pin in high-impedance state.

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_POE3_Stop

Disable the pin in high-impedance state.

[Syntax]

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

[Argument(s)]
None.

[Return value]
None.
**R_POE3_Set_HiZ_MTUₘ**

Places the MTUₘ pin in high-impedance state. Some value of m may cause MTU pins or GPT pins to be in high-impedance state.

**[Syntax]**

```c
void R_POE3_Set_HiZ_MTU_m( void );
```

**Remark**  
ₘ is the channel number. ₘ can contain 0, 3_4, 6_7 as the value. In case of ₘ=3_4, it specifies both MTU3 and MTU4 or GPT0~GPT2. In case of ₘ=6_7, it specifies both MTU6 and MTU7.

**[Argument(s)]**

None.

**[Return value]**

None.
Disables the MTU\textsubscript{m} pin in high-impedance state. Some value of \textit{m} may cause MTU pins or GPT pins in high-impedance state to be disabled.

### Syntax

```c
void R_POE3_Clear_HiZ_MTU\textsubscript{m}( void );
```

### Remark

\textit{m} is the channel number. \textit{m} can contain 0, 3-4, 6-7 as the value. In case of \textit{m}=3,4, it specifies both MTU3 and MTU4 or GPT0~GPT2. In case of \textit{m}=6,7, it specifies both MTU6 and MTU7.

### Argument(s)

None.

### Return value

None.
R_POE3_Set_HiZ_GPT3

Places the GPT3 pin in high-impedance state.

[Syntax]

```c
void R_POE3_Set_HiZ_GPT3( void );
```

[Argument(s)]

None.

[Return value]

None.
Disables the GPT3 pin in high-impedance state.

[Syntax]

```c
void R_POE3_Clear_HiZ_GPT3( void );
```

[Argument(s)]
None.

[Return value]
None.
### 3.2.10 General PWM Timer

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_GPT_Create</td>
<td>Performs initialization required to control the general PWM timer.</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_etgin_interrupt</td>
<td>Performs processing in response to the interrupt source ETGIN (External trigger falling input).</td>
</tr>
<tr>
<td>r_gpt_etgip_interrupt</td>
<td>Performs processing in response to the interrupt source ETGIP (External trigger rising input).</td>
</tr>
<tr>
<td>r_gpt_gtcian_interrupt</td>
<td>Performs processing in response to the interrupt source GTCIAn (GPTn.GTCCRA input capture/compare match).</td>
</tr>
<tr>
<td>r_gpt_gtcibn_interrupt</td>
<td>Performs processing in response to the interrupt source GTCIBn (GPTn.GTCCRB input capture/compare match).</td>
</tr>
<tr>
<td>r_gpt_gtcicn_interrupt</td>
<td>Performs processing in response to the interrupt source GTCICn (GPTn.GTCCRC compare match).</td>
</tr>
<tr>
<td>r_gpt_gtcidn_interrupt</td>
<td>Performs processing in response to the interrupt source GTCIDn (GPTn.GTCCRD compare match).</td>
</tr>
<tr>
<td>r_gpt_gtcien_interrupt</td>
<td>Performs processing in response to the interrupt source GTCIEn (GPTn.GTCCRE compare match).</td>
</tr>
<tr>
<td>r_gpt_gtcifn_interrupt</td>
<td>Performs processing in response to the interrupt source GTClFn (GPTn.GTCCRF compare match).</td>
</tr>
<tr>
<td>r_gpt_gdten_interrupt</td>
<td>Performs processing in response to the interrupt source GDTEn (dead time error).</td>
</tr>
<tr>
<td>r_gpt_gtcivn_interrupt</td>
<td>Performs processing in response to the interrupt source GTCIVn (GPTn.GTCNT overflow or GPTn.GTPR compare match).</td>
</tr>
<tr>
<td>r_gpt_gtcinvn_interrupt</td>
<td>Performs processing in response to the interrupt source GTCIUn (GPTn.GTCNT underflow).</td>
</tr>
<tr>
<td>R_GPTn_Start</td>
<td>Starts GPTn.GTCNT count.</td>
</tr>
<tr>
<td>R_GPTn_Stop</td>
<td>Ends GPTn.GTCNT count.</td>
</tr>
<tr>
<td>R_GPTn_HardwareStart</td>
<td>Enables the GPTn interrupt(except the interrupt source GDTEn (dead time error)).</td>
</tr>
<tr>
<td>R_GPTn_HardwareStop</td>
<td>Disables the GPTn interrupt(except the interrupt source GDTEn (dead time error)).</td>
</tr>
</tbody>
</table>
**R_GPT_Create**

Performs initialization required to control the general PWM timer.

**[Syntax]**

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

**[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_etgin_interrupt

Performs processing in response to the interrupt source ETGIN (External trigger falling input).

[Syntax]

```c
void r_gpt_etgin_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.
r_gpt_etgip_interrupt

Performs processing in response to the interrupt source ETGIP (External trigger rising input).

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
**r_gpt_gtcian_interrupt**

Performs processing in response to the interrupt source GTCIA\(n\) (GPT\(n\).GTCCRA input capture/compare match).

**[Syntax]**

```c
void r_gpt_gtcian_interrupt ( void );
```

Remark

\(n\) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
Performs processing in response to the interrupt source GTCIBn (GPTn.GTCCRB input capture/compare match).

**Syntax**

```c
void r_gpt_gtcibn_interrupt ( void );
```

Remark

\( n \) is the channel number.

**Argument(s)**

None.

**Return value**

None.
r_gpt_gtcicn_interrupt

Performs processing in response to the interrupt source GTCIC\(n\) (GPT\(n\).GTCCRC compare match).

[Syntax]

```c
void r_gpt_gtcicn_interrupt ( void );
```

Remark \(n\) is the channel number.

[Argument(s)]

None.

[Return value]

None.
r_gpt_gtcidn_interrupt

Performs processing in response to the interrupt source GTCIDn (GPTn.GTCCRD compare match).

[Syntax]

```c
void r_gpt_gtcidn_interrupt ( void );
```

Remark    

$n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
r_gpt_gtcien_interrupt

Performs processing in response to the interrupt source GTCIE\textsubscript{n} (GPT\textsubscript{n}.GTCCRE compare match).

**[Syntax]**

```c
void r_gpt_gtcien_interrupt ( void );
```

Remark  \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.


### r_gpt_gtcifn_interrupt

Performs processing in response to the interrupt source GTCIFn (GPTn.GTCCRF compare match).

**[Syntax]**

```c
void r_gpt_gtcifn_interrupt ( void );
```

**Remark**

\( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_gpt_gdten_interrupt**

Performs processing in response to the interrupt source GDTE\textsubscript{n} (dead time error).

**[Syntax]**

```c
void r_gpt_gdten_interrupt ( void );
```

**Remark**

\( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_gpt_gtcivn_interrupt

Performs processing in response to the interrupt source GTCIV\textsubscript{n} (GPT\textsubscript{n}.GTCNT overflow or GPT\textsubscript{n}.GTPR compare match).

**[Syntax]**

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

Remark \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_gpt_gtciun_interrupt**

Performs processing in response to the interrupt source GTCIU\(n\) (GPT\(n\).GTCNT underflow).

**[Syntax]**

```c
void r_gpt_gtciun_interrupt ( void );
```

**Remark**

\(n\) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_GPTn_Start

Starts GPTn.GTCNT counter.

[Syntax]

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

Remark  \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_GPTn_Stop

Ends GPTn.GTCNT counter.

[Syntax]

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

Remark  
$n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_GPTn_HardwareStart

Enables the GPTn interrupt (except the interrupt source GDTEn (dead time error)).

Enables the GPTn interrupt of the following interrupt sources.

<table>
<thead>
<tr>
<th>GPT interrupt sources</th>
<th>GPTn interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTCIA[n]</td>
<td>GPTn.GTCCRA input capture/compare match</td>
</tr>
<tr>
<td></td>
<td>Channel[n] input capture/compare match A interrupt</td>
</tr>
<tr>
<td>GTCIB[n]</td>
<td>GPTn.GTCCRB input capture/compare match</td>
</tr>
<tr>
<td></td>
<td>Channel[n] input capture/compare match B interrupt</td>
</tr>
<tr>
<td>GTCIC[n]</td>
<td>GPTn.GTCCRC compare match</td>
</tr>
<tr>
<td></td>
<td>Channel[n] compare match C interrupt</td>
</tr>
<tr>
<td>GTCID[n]</td>
<td>GPTn.GTCCRD compare match</td>
</tr>
<tr>
<td></td>
<td>Channel[n] compare match D interrupt</td>
</tr>
<tr>
<td>GTCIE[n]</td>
<td>GPTn.GTCCRE compare match</td>
</tr>
<tr>
<td></td>
<td>Channel[n] compare match E interrupt</td>
</tr>
<tr>
<td>GTCIF[n]</td>
<td>GPTn.GTCCRF compare match</td>
</tr>
<tr>
<td></td>
<td>Channel[n] compare match F interrupt</td>
</tr>
<tr>
<td>GTCI[U]n</td>
<td>GPTn.GTCNT overflow</td>
</tr>
<tr>
<td></td>
<td>Channel[n] overflow interrupt</td>
</tr>
<tr>
<td></td>
<td>GPTn.GTCNT underflow</td>
</tr>
<tr>
<td></td>
<td>Channel[n] underflow interrupt</td>
</tr>
</tbody>
</table>

**[Syntax]**

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

Remark  
\[n\] is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
## R_GPTn_HardwareStop

Disables the GPT\(n\) interrupt (except the interrupt source GDTE\(n\) (dead time error)).

Disables the GPT\(n\) interrupt of the following interrupt sources.

<table>
<thead>
<tr>
<th>GPT interrupt sources</th>
<th>GPT(n) interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTCIA(n)</td>
<td>GPT(n).GTCCRA input capture/compare match</td>
</tr>
<tr>
<td>GTCIB(n)</td>
<td>GPT(n).GTCCR(B) input capture/compare match</td>
</tr>
<tr>
<td>GTCIC(n)</td>
<td>GPT(n).GTCCRC compare match</td>
</tr>
<tr>
<td>GTCID(n)</td>
<td>GPT(n).GTCCRD compare match</td>
</tr>
<tr>
<td>GTCIE(n)</td>
<td>GPT(n).GTCCR(E) compare match</td>
</tr>
<tr>
<td>GTCIF(n)</td>
<td>GPT(n).GTCCRF compare match</td>
</tr>
<tr>
<td>GTCIV(n)</td>
<td>GPT(n).GTCCNT overflow</td>
</tr>
<tr>
<td>GTCIU(n)</td>
<td>GPT(n).GTCCNT underflow</td>
</tr>
</tbody>
</table>

### [Syntax]

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

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

### [Argument(s)]

None.

### [Return value]

None.
3.2.11 16-Bit Timer Pulse Unit

Below is a list of API functions output by the Code Generator for 16-Bit Timer Pulse Unit use.

Table 3.12 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 required to control the 16-bit timer pulse unit.</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_tgina_interrupt</td>
<td>Performs processing in response to the interrupt source TGI\textsubscript{A} (TPUn.TGRA input capture/compare match).</td>
</tr>
<tr>
<td>r_tpu_tginb_interrupt</td>
<td>Performs processing in response to the interrupt source TGI\textsubscript{B} (TPUn.TGRB input capture/compare match).</td>
</tr>
<tr>
<td>r_tpu_tginc_interrupt</td>
<td>Performs processing in response to the interrupt source TGI\textsubscript{C} (TPUn.TGRC input capture/compare match).</td>
</tr>
<tr>
<td>r_tpu_tgind_interrupt</td>
<td>Performs processing in response to the interrupt source TGI\textsubscript{D} (TPUn.TGRD input capture/compare match).</td>
</tr>
<tr>
<td>r_tpu_tcinv_interrupt</td>
<td>Performs processing in response to the interrupt source TCIn\textsubscript{V} (TPUn.TCNT overflow).</td>
</tr>
<tr>
<td>r_tpu_tcinu_interrupt</td>
<td>Performs processing in response to the interrupt source TCIn\textsubscript{U} (TPUn.TCNT underflow).</td>
</tr>
<tr>
<td>R_TPUn_Start</td>
<td>Starts TCNT count of TPUn.</td>
</tr>
<tr>
<td>R_TPUn_Stop</td>
<td>Ends TCNT count of TPUn.</td>
</tr>
</tbody>
</table>
R_TPU_Create

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

[Syntax]

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

[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]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
3. GENERAL

**r_tpu_tgina_interrupt**

Performs processing in response to the interrupt source TGInA (TPUn.TGRA input capture/compare match).

**[Syntax]**

```c
void r_tpu_tgina_interrupt ( void );
```

Remark  

$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_tpu_tginb_interrupt

Performs processing in response to the interrupt source TGI\(n\)B (TPUn.TGRB input capture/compare match).

**[Syntax]**

```c
void r_tpu_tginb_interrupt ( void );
```

**Remark**

\(n\) is the channel number.

**[Argument(s)]**

None.

**[Return Value]**

None.
r_tpu_tginc_interrupt

Performs processing in response to the interrupt source TGInC (TPU.n.TGRC input capture/compare match).

[Syntax]

```c
void r_tpu_tginc_interrupt ( void );
```

Remark  
$n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
r_tpu_tgind_interrupt

Performs processing in response to the interrupt source TGInD (TPUn.TGRD input capture/compare match).

[Syntax]

```c
void r_tpu_tgind_interrupt ( void );
```

Remark

- \( n \) is the channel number.

[Argument(s)]

- None.

[Return value]

- None.
r_tpu_tciNv_interrupt

Performs processing in response to the interrupt source TCInV (TPUn.TCNT overflow).

[Syntax]

```c
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 interrupt source TCInU (TPUn.TCNT underflow).

**[Syntax]**

```c
void r_tpu_tcinu_interrupt ( void );
```

**Remark**

\( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_TPU\_n\_Start

Starts TCNT count of TPU\_n.

[Syntax]

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

Remark \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
**R_TPU\(_n\)_Stop**

Ends TCNT count of TPU\(_n\).

### [Syntax]

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

**Remark**

\( n \) is the channel number.

### [Argument(s)]

None.

### [Return value]

None.
3.2.12 Programmable Pulse Generator

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

Table 3.13 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 required 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 required to control the programmable pulse generator.

#### 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.
3.2.13 Compare Match Timer

Below is a list of API functions output by the Code Generator for Compare Match Timer use.

Table 3.14 API Functions: [Compare Match Timer]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_CMTn_Create</td>
<td>Performs initialization required to control the compare match timer.</td>
</tr>
<tr>
<td>R_CMTn_Create_UserInit</td>
<td>Performs user-defined initialization relating to the compare match timer.</td>
</tr>
<tr>
<td>r_cmt_cmin_interrupt</td>
<td>Performs processing in response to the interrupt source CMin (CMCNTn counter and CMCORn register compare match).</td>
</tr>
<tr>
<td>R_CMTn_Start</td>
<td>Starts CMCNTn count.</td>
</tr>
<tr>
<td>R_CMTn_Stop</td>
<td>Ends CMCNTn count.</td>
</tr>
</tbody>
</table>
R_CMTn_Create

Performs initialization required to control the compare match timer.

[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_cmn_interrupt

Performs processing in response to the interrupt source CMI\textsubscript{n} (CMCNT\textsubscript{n} counter and CMCOR\textsubscript{n} register compare match).

**Syntax**

```
__interrupt static void r_cmt_cmn_interrupt ( void );
```

Remark \( n \) is the channel number.

**Argument(s)**

None.

**Return value**

None.
**R_CMTn_Start**

Starts CMCNTn count.

**[Syntax]**

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

Remark  
$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_CMT\textit{n}_Stop

Ends CMCNT\textit{n} count.

[Syntax]

\begin{verbatim}
void R_CMT\textit{n}_Stop ( void );
\end{verbatim}

Remark \textit{n} is the channel number.

[Argument(s)]

None.

[Return value]

None.
3.2.14 Compare Match Timer W

Below is a list of API functions output by the Code Generator for Compare Match Timer W use.

Table 3.15 API Functions: [Compare Match Timer W]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_CMTWm_Create</td>
<td>Performs initialization required to control the compare match timer W.</td>
</tr>
<tr>
<td>R_CMTWm_Create_UserInit</td>
<td>Performs user-defined initialization relating to the compare match timer W.</td>
</tr>
<tr>
<td>r_cmtw_cnwim_interrupt</td>
<td>Performs processing in response to the interrupt source CMWI (compare match).</td>
</tr>
<tr>
<td>r_cmtw_icnim_interrupt</td>
<td>Performs processing in response to the interrupt source ICnlm (output compare).</td>
</tr>
<tr>
<td>r_cmtw_ocnim_interrupt</td>
<td>Performs processing in response to the interrupt source OCnlm (output compare).</td>
</tr>
<tr>
<td>R_CMTWm_Start</td>
<td>Starts CMWCNTm count.</td>
</tr>
<tr>
<td>R_CMTWm_Stop</td>
<td>Ends CMWCNTm count.</td>
</tr>
</tbody>
</table>
R_CMTWm_Create

Performs initialization required to control the compare match timer W.

[Syntax]

```c
void R_CMTWm_Create ( void );
```

Remark  \( m \) is the unit number.

[Argument(s)]
None.

[Return value]
None.
R_CMTWm_Create_UserInit

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

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

[Syntax]

```c
void R_CMTWm_Create_UserInit ( void );
```

Remark  \( m \) is the unit number.

[Argument(s)]

None.

[Return value]

None.
r_cmtw_cmwim_interrupt

Performs processing in response to the interrupt source CMWI (compare match).

[Syntax]

```c
void r_cmtw_cmwim_interrupt ( void );
```

Remark $m$ is the unit number.

[Argument(s)]

None.

[Return value]

None.
r_cmtw_icnim_interrupt

Performs processing in response to the interrupt source ICnm (input compare).

[Syntax]

```c
void r_cmtw_icnim_interrupt ( void );
```

Remark  $m$ is the unit number.

[Argument(s)]

None.

[Return value]

None.
AP4  3. GENERAL

r_cmtw_ocnim_interrupt

Performs processing in response to the interrupt source OCnIm (output comparer).

[Syntax]

void r_cmtw_ocnim_interrupt ( void );

Remark  \( n \) is the interrupt number, \( m \) is the unit number.

[Argument(s)]

None.

[Return value]

None.
R_CMTWm_Start

Starts CMWCNTm count.

**[Syntax]**

```c
void R_CMTWm_Start ( void );
```

Remark  

$m$ is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_CMTWm_Stop

Ends CMWCNT$m count.

[Syntax]

```c
void R_CMTWm_Stop ( void );
```

Remark $m$ is the unit number.

[Argument(s)]

None.

[Return value]

None.
3.2.15 Watchdog Timer

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

Table 3.16 API Functions: [Watchdog Timer]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_WDTn_Create</td>
<td>Performs initialization required to control the watchdog timer.</td>
</tr>
<tr>
<td>R_WDTn_Create_UserInit</td>
<td>Performs user-defined initialization relating to the watchdog timer.</td>
</tr>
<tr>
<td>r_wdtndef_refef_interrupt</td>
<td>Performs processing in response to the WDT overflow / refresh error.</td>
</tr>
<tr>
<td>R_WDTn_Restart</td>
<td>Clears the watchdog timer counter and resumes counting.</td>
</tr>
</tbody>
</table>
R_WDTn_Create

Performs initialization required to control the watchdog timer.

[Syntax]

```c
void R_WDTn_Create ( void );
```

Remark  

$n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
**R_WDTn_Create_UserInit**

Performs user-defined initialization relating to the watchdog timer.

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

**[Syntax]**

```c
void R_WDTn_Create_UserInit ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_wdt_n_undff_refef_interrupt

Performs processing in response to the WDT overflow / refresh error.

[Syntax]

```c
void r_wdt_n_undff_refef_interrupt ( void );
```

Remark  
$n$ is the channel number.

[Argument(s)]
None.

[Return value]
None.
R_WDTn_Restart

Clears the watchdog timer counter and resumes counting.

[Syntax]

```c
void R_WDTn_Restart ( void );
```

Remark

\( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
3.2.16 Independent Watchdog Timer

Below is a list of API functions output by the Code Generator for Independent Watchdog Timer use.

Table 3.17 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 required 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_undff_refef_interrupt</td>
<td>Performs processing in response to the IWDT overflow / refresh error.</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 required to control the independent watchdog timer.

**[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_undff_reffef_interrupt

Performs processing in response to the IWDT overflow / refresh error.

[Syntax]

void  r_iwdt_undff_reffef_interrupt ( void );

Remark   n is the channel number.

[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.
3.2.17 Serial Communications Interface with FIFO

Below is a list of API functions output by the Code Generator for Serial Communication Interface with FIFO use.

Table 3.18 API Functions: [Serial Communication Interface with FIFO]

<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 serial communications interface with FIFO.</td>
</tr>
<tr>
<td>R_SCIFAn_Create_UserInit</td>
<td>Performs user-defined initialization relating to the serial communications interface with FIFO.</td>
</tr>
<tr>
<td>r_scifan_txifn_interrupt</td>
<td>Performs processing in response to the interrupt source TXI (FIFO data empty interrupts).</td>
</tr>
<tr>
<td>r_scifan_rxifn_interrupt</td>
<td>Performs processing in response to the interrupt source RXI (receive FIFO data full interrupts).</td>
</tr>
<tr>
<td>r_scifan_brifn_interrupt</td>
<td>Performs processing in response to the interrupt source BRI (break or overrun interrupts) and ERI (framing error or parity error interrupts).</td>
</tr>
<tr>
<td>r_scifan_drifn_interrupt</td>
<td>Performs processing in response to the interrupt source DRI (receive data ready interrupts) and TEI (transmit-end 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>
<tr>
<td>R_SCIFAn_Start</td>
<td>Starts serial communications interface with FIFO.</td>
</tr>
<tr>
<td>R_SCIFAn_Stop</td>
<td>Ends serial communications interface with FIFO.</td>
</tr>
<tr>
<td>R_SCIFAn_Serial_Send</td>
<td>Starts serial communications interface with FIFO transmission (asynchronous mode).</td>
</tr>
<tr>
<td>R_SCIFAn_Serial_Receive</td>
<td>Starts serial communications interface with FIFO reception (asynchronous mode).</td>
</tr>
<tr>
<td>R_SCIFAn_Serial_Send_Receive</td>
<td>Starts serial communications interface with FIFO transmission/reception (clock synchronous mode).</td>
</tr>
</tbody>
</table>
R_SCIFAn_Create

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

[Syntax]

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

Remark      \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_SCIFAn_Create_UserInit

Performs user-defined initialization relating to the serial communications interface with FIFO.

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

[Syntax]

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

Remark  \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
r_scifan_txfn_interrupt

Performs processing in response to the interrupt source TXI (FIFO data empty interrupts).

[Syntax]

```c
void r_scifan_txfn_interrupt ( void );
```

Remark

\( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
Performs processing in response to the interrupt source RXI (receive FIFO data full interrupts).

**Syntax**

```c
void r_scifan_rxifn_interrupt ( void );
```

**Remark**

`n` is the channel number.

**Argument(s)**

None.

**Return value**

None.
**r_scifan_brfn_interrupt**

Performs processing in response to the interrupt source BRI (break or overrun interrupts) and ERI (framing error or parity error interrupts).

**[Syntax]**

```c
void r_scifan_brfn_interrupt ( void );
```

Remark  

\( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_scifan_drifn_interrupt**

Performs processing in response to the interrupt source DRI (receive data ready interrupts) and TEI (transmit-end interrupts).

**Syntax**

```c
void r_scifan_drifn_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_teifn_interrupt callback routine.

**[Syntax]**

```c
void r_scifan_callback_transmitend ( void );
```

Remark  

n is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
Performs processing in response to the receive FIFO data full interrupts.

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

**r_scifan_callback_receiveend**

**Syntax**

```c
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.

Remarks: This API function is called as the r_scifan_erifn_interrupt or r_scifan_brifn_interrupt callback routine.

[Bias]

```c
void r_scifan_callback_error ( void );
```

Remarks: \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_SCIFAn_Start

Starts serial communications interface with FIFO.

[Syntax]

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

Remark

$n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_SCIFAn_Stop

Ends serial communications interface with FIFO.

**[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).

Remarks 1. This API function repeats the byte-level SCI transmission from the buffer specified in argument \texttt{txbuf} the number of times specified in argument \texttt{txnum}.

Remarks 2. When performing a SCI transmission, \texttt{R_SCIFAn_Start} must be called before this API function is called.

[Syntax]

```c
MD_STATUS R_SCIFAn_Serial_Send ( const uint8_t * txbuf, uint16_t txnum );
```

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>const uint8_t * txbuf;</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t txnum;</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).

Remarks 1. This API function repeats the byte-level SCI reception from the buffer specified in argument \textit{rxbuf} the number of times specified in argument \textit{rxnum}.

Remarks 2. When performing a SCI reception, \texttt{R_SCIFAn_Start} must be called before this API function is called.

[Syntax]

```
MD_STATUS    R_SCIFAn_Serial_Receive ( uint8_t * rxbuf, uint16_t rxnum );
```

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>O</td>
<td>uint8_t * rxbuf;</td>
<td>Pointer to a buffer to store the reception data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t rxnum;</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_Serial_Send_Receive

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

Remarks 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`.

Remarks 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`.

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

[Syntax]

```c
MD_STATUS R_SCIFAn_Serial_Send_Receive(const uint8_t * tx_buf, uint16_t tx_num, uint8_t * 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>const uint8_t * 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 * 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>
### 3.2.18 I²C Bus Interface

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

#### Table 3.19 API Functions: [I²C Bus Interface]

<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. Generates Stop condition when transmission is completed.</td>
</tr>
<tr>
<td>R_RIICn_Master_Send_Without_Stop</td>
<td>Starts RIIC master transmission. Not generates Stop condition when transmission is completed.</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.

[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

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

Remark  \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
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_riicn_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]

```c
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.

Remarks 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`.

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

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

#### [Syntax]

```c
MD_STATUS R_RIICn_Master_Send(uint16_t adr, const uint8_t * 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><code>uint16_t *adr;</code></td>
<td>Slave address</td>
</tr>
<tr>
<td>I</td>
<td><code>const uint8_t *tx_buf;</code></td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td><code>uint16_t tx_num;</code></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 <code>adr</code> specification</td>
</tr>
</tbody>
</table>
**R_RIICn_Master_Send_Without_Stop**

Starts RIIC master transmission. Not generates Stop condition when transmission is completed.

Remarks 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`.

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

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

**[Syntax]**

```c
MD_STATUS R_RIICn_Master_Send_Without_Stop(uint16_t adr, const uint8_t * 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 <code>adr</code></td>
<td>Slave address</td>
</tr>
<tr>
<td>I</td>
<td>const uint8_t * <code>tx_buf</code></td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t <code>tx_num</code></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 <code>adr</code> specification</td>
</tr>
</tbody>
</table>
R_RIICn_Master_Receive

Starts RIIC master reception.

Remarks 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`.

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

Remarks 3. When performing a RIIC master reception, `R_RIICn_Start` must be called before this API function is called.

[Syntax]

```c
MD_STATUS R_RIICn_Master_Receive(uint16_t adr, uint8_t * 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</td>
<td><code>adr</code>; Slave address</td>
</tr>
<tr>
<td>O</td>
<td>uint8_t *</td>
<td><code>rx_buf</code>; Pointer to a buffer to store the reception data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t</td>
<td><code>rx_num</code>; 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 <code>adr</code> specification</td>
</tr>
</tbody>
</table>
**R_RIIcn_Slave_Send**

Starts RIIC slave transmission.

Remarks 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`.

Remarks 2. When performing a RIIC slave transmission, `R_RIIcn_Start` must be called before this API function is called.

**[Syntax]**

```c
void R_RIIcn_Slave_Send ( const uint8_t * 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>const uint8_t * 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]**

None.
### R_RIICn_Slave_Receive

Starts RIIC slave reception.

**Remarks 1.** This API function performs byte-level RIIC slave reception the number of times specified by the argument `rx_num`, and stores the data in the buffer specified by the argument `rx_buf`.

**Remarks 2.** When performing a RIIC slave reception, `R_RIICn_Start` must be called before this API function is called.

#### [Syntax]

```c
void R_RIICn_Slave_Receive ( uint8_t * 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><code>uint8_t * rx_buf</code></td>
<td>Pointer to a buffer to store the reception data</td>
</tr>
<tr>
<td>I</td>
<td><code>uint16_t rx_num</code></td>
<td>Total amount of data to receive</td>
</tr>
</tbody>
</table>

#### [Return value]

- None.
### R_RIICn_StartCondition

Issues the start condition and causes a transfer error and an event generation interrupt (EEI).

Remarks 1. This API function is called as the internal function of R_RIICn_Master_Send and R_RIICn_Master_Receive.

Remarks 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_ricn_error_interrupt is called in response to calling of this API function.

**Syntax**

```c
void R_RIICn_StopCondition ( void );
```

Remark $n$ is the channel number.

**Argument(s)**

None.

**Return value**

None.
**r_riicn_callback_receiveerror**

Of the internal processing for transfer errors and event generation interrupts (EEI), this function handles processing specialized in the arbitration-lost detection, NACK detection, and timeout detection.

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

**[Syntax]**

```
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</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_ERROR1: Arbitration-lost detection</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_ERROR2: Timeout detection</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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.
r_riic\textit{n} \_callback \_receiveend

Of the internal processing for transfer errors and event generation interrupts (EEI), this function handles processing specialized in the start condition detection in response to calling of \texttt{R\textunderscore RII\textit{Cn} \_Master \_Receive}.

Remark This API function is called as the \texttt{r\textunderscore riic\textit{n}\_error\_interrupt} callback routine.

[Syntax]

\begin{verbatim}
  static void r_riic\textit{n}\_callback\_receiveend ( void );
\end{verbatim}

Remark \textit{n} is the channel number.

[Argument(s)]

None.

[Return value]

None.
### 3.2.19 Serial Peripheral Interface

Below is a list of API functions output by the Code Generator for Serial Peripheral Interface use.

Table 3.20 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\_Create**

Performs initialization necessary to control the serial peripheral interface.

**[Syntax]**

```c
void R_RSPI\_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.
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
void r_rspin_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]

```
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
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
void r_rspin_idle_interrupt ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_RSPI_n_Start

Starts RSPI communication.

[Syntax]

```c
void R_RSPI_n_Start ( void );
```

Remark $n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_RSPI_Stop

Ends RSPI communication.

[Syntax]

```c
void R_RSPI_Stop ( void );
```

Remark    $n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
### R_RSPI\_n\_Send

Starts RSPI transmission.

Remarks 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*.

Remarks 2. When performing a RSPI transmission, **R_RSPI\_Start** must be called before this API function is called.

[Syntax]

```c
MD_STATUS R_RSPI\_n\_Send (const uint32_t * 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>const uint32_t * 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 <em>tx_num</em> specification</td>
</tr>
</tbody>
</table>

R_RSPIn_Send_Receive

Starts RSPI transmission/reception.

Remarks 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`.

Remarks 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`.

Remarks 3. When performing a RSPI transmission/reception, `R_RSPIn_Start` must be called before this API function is called.

[Syntax]

```c
MD_STATUS R_RSPIn_Send_Receive (const uint32_t * tx_buf, uint16_t tx_num, uint32_t * 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>const uint32_t * 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 * 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 <code>tx_num</code></td>
</tr>
</tbody>
</table>
**r_rspin_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
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><code>uint8_t err_type;</code></td>
<td>Source of the RSPI error interrupt (x is undefined)</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>xxxx00x1B: Overrun error detection</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>xxxx01x0B: Mode fault error detection</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>xxxx10x0B: Parity error detection</code></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.
### 3.2.20 SPI Multi I/O Bus Controller

Below is a list of API functions output by the Code Generator for SPI Multi I/O Bus Controller use.

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_SPIBSC_Create</td>
<td>Performs initialization required to control the SPI multi I/O bus controller.</td>
</tr>
<tr>
<td>R_SPIBSC_Create_UserInit</td>
<td>Performs user-defined initialization relating to the SPI multi I/O bus controller.</td>
</tr>
<tr>
<td>R_SPIBSC_EAVUpperAddressChange</td>
<td>Changes the address of higher 8-bit in the address 32-bit output.</td>
</tr>
<tr>
<td>R_SPIBSC_SPIRead</td>
<td>Receives data.</td>
</tr>
<tr>
<td>R_SPIBSC_SPIWrite</td>
<td>Transmits data.</td>
</tr>
<tr>
<td>R_SPIBSC_SPIRead_Write</td>
<td>Receives and transmits data.</td>
</tr>
</tbody>
</table>
R_SPIBSC_Create

Performs initialization required to control the SPI multi I/O bus controller.

[[Syntax]]

```
void R_SPIBSC_Create ( void );
```

[[Argument(s)]]

None.

[[Return value]]

None.
**R_SPIBSC_Create_UserInit**

Performs user-defined initialization relating to the SPI multi I/O bus controller.

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

**[Syntax]**

```c
void R_SPIBSC_Create_UserInit ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_SPIBSC_EAVUpperAddressChange

Changes the address of higher 8-bit in the address 32-bit output.

[Syntax]

```c
void R_SPIBSC_EAVUpperAddressChange(uint8_t const up_adr);
```

[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 up_adr;</td>
<td>Address of higher 8-bit in the address 32-bit output</td>
</tr>
</tbody>
</table>

[Return value]

None.
**R_SPIBSC_SPIRead**

Receives the data with the SPI-mode.

**[Syntax]**

```c
MD_STATUS R_SPIBSC_SPIRead(st_spibsc_spiparam const param, uint32_t rx_data);
```

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>st_spibsc_spiparam const param;</td>
<td>Parameter of the SPI-mode (Read / Write)</td>
</tr>
<tr>
<td>O</td>
<td>uint32_t rx_data;</td>
<td>Received 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 <code>param</code> specification</td>
</tr>
</tbody>
</table>
**R_SPIBSC_SPIWrite**

Transmits the data with the SPI-mode.

**[Syntax]**

```c
MD_STATUS R_SPIBSC_SPIWrite(st_spibsc_spiparam const param, uint32_t const tx_data);
```

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>st_spibsc_spiparam const param;</td>
<td>Parameter of the SPI-mode (Read / Write)</td>
</tr>
<tr>
<td>I</td>
<td>uint32_t const tx_data;</td>
<td>Transmitted 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 <code>param</code> specification</td>
</tr>
</tbody>
</table>
R_SPIBSC_SPIRead_Write

Receives and transmits the data with the SPI-mode.

[Syntax]

```
MD_STATUS R_SPIBSC_SPIRead_Write(st_spibsc_spiparam const param, uint32_t rx_data, uint32_t const tx_data);
```

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>st_spibsc_spiparam const param;</td>
<td>Parameter of the SPI-mode (Read / Write)</td>
</tr>
<tr>
<td>O</td>
<td>uint32_t const rx_data;</td>
<td>Received data</td>
</tr>
<tr>
<td>I</td>
<td>uint32_t const tx_data;</td>
<td>Transmitted 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 param specification</td>
</tr>
</tbody>
</table>

```
3.2.21 CRC Operation Units

Below is a list of API functions output by the Code Generator for CRC Operation Units use.

Table 3.22 API Functions: [CRC Operation Units]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_CRC_SetCRC8_2F</td>
<td>Initializes the CRC operation unit for 8-bit CRC calculation (CCRC8-0x2F).</td>
</tr>
<tr>
<td>R_CRC_SetCRC8_SAE</td>
<td>Initializes the CRC operation unit for 8-bit CRC calculation (CRC8-SAE-J1850).</td>
</tr>
<tr>
<td>R_CRC_SetCRC16_CCITT</td>
<td>Initializes the CRC operation unit for 16-bit CRC calculation (CRC16-CCITT).</td>
</tr>
<tr>
<td>R_CRC_SetCRC32_ETHER</td>
<td>Initializes the CRC operation unit for 32-bit CRC calculation (CRC32_ETHERNET).</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_2F

Initializes the CRC operation unit for 8-bit CRC calculation (CCRC8-0x2F).

[Syntax]

```c
void R_CRC_SetCRC8_2F ( void );
```

[Argument(s)]

None.

[Return value]

None.
**R_CRC_SetCRC8_SAE**

Initializes the CRC operation unit for 8-bit CRC calculation (CRC8-SAE-J1850).

**[Syntax]**

```c
void R_CRC_SetCRC8_SAE ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
3. GENERAL

R_CRC_SetCRC16_CCITT

Initializes the CRC operation unit for 16-bit CRC calculation (CRC16-CCITT).

[Syntax]

```c
void R_CRC_SetCRC16_CCITT ( void );
```

[Argument(s)]

None.

[Return value]

None.
R_CRC_SetCRC32_ETHER

Initializes the CRC operation unit for 32-bit CRC calculation (CRC32_ETHERNET).

[Syntax]

```c
void R_CRC_SetCRC32_ETHER ( 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 ( 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>uint32_t data;</td>
<td>The initial value of the data from which the CRC is to be calculated</td>
</tr>
</tbody>
</table>

#### [Return value]

None.
R_CRC_Get_Result

Gets the result of operation.

[Syntax]

```c
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>uint32_t * const result;</td>
<td>Pointer to the location where the result of operation is stored</td>
</tr>
</tbody>
</table>

[Return value]

None.
### 3.2.22 ΔΣ Interface

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

Table 3.23 API Functions: [ΔΣ Interface]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_DSMIF_Create</td>
<td>Performs initialization required to control the ΔΣ interface.</td>
</tr>
<tr>
<td>R_DSMIF_Create_UserInit</td>
<td>Performs user-defined initialization relating to the ΔΣ interface.</td>
</tr>
<tr>
<td>R_DSMIF_UVW_Start</td>
<td>Filtering is started of ΔΣ interface (unit0 (U, V, W)).</td>
</tr>
<tr>
<td>R_DSMIF_UVW_Stop</td>
<td>Filtering is stopped of ΔΣ interface (unit0 (U, V, W)).</td>
</tr>
<tr>
<td>R_DSMIF_X_Start</td>
<td>Filtering is started of ΔΣ interface (unit1 (X)).</td>
</tr>
<tr>
<td>R_DSMIF_X_Stop</td>
<td>Filtering is stopped of ΔΣ interface (unit1 (X)).</td>
</tr>
<tr>
<td>r_dsmif_uvw_overcurrent_interrupt</td>
<td>Performs processing in response to the overcurrent interrupt of ΔΣ interface (unit0 (U, V, W)).</td>
</tr>
<tr>
<td>r_dsmif_uvw_totalcurrent_interrupt</td>
<td>Performs processing in response to the totalcurrent interrupt of ΔΣ interface (unit0 (U, V, W)).</td>
</tr>
<tr>
<td>r_dsmif_uvw_shortcircuit_interrupt</td>
<td>Performs processing in response to the short circuit interrupt of ΔΣ interface (unit0 (U, V, W)).</td>
</tr>
<tr>
<td>r_dsmif_x_overcurrent_interrupt</td>
<td>Performs processing in response to the overcurrent interrupt of ΔΣ interface (unit1 (X)).</td>
</tr>
<tr>
<td>r_dsmif_x_shortcircuit_interrupt</td>
<td>Performs processing in response to the short circuit interrupt of ΔΣ interface (unit1 (X)).</td>
</tr>
</tbody>
</table>
R_DSMIF_Create

Performs initialization required to control the ΔΣ interface.

[Syntax]

```c
void R_DSMIF_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.
R_DSMIF_Create_UserInit

Performs user-defined initialization relating to the ΔΣ interface.

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

[Syntax]

```
void R_DSMIF_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.
**R_DSMIF_UVW_Start**

Filtering is started of ΔΣ interface (unit0 (U, V, W)).

**[Syntax]**

```c
void R_DSMIF_UVW_Start ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
Filtering is stopped of ΔΣ interface (unit0 (U, V, W)).

[Syntax]

```c
void R_DSMIF_UVW_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.
Filtering is started of ΔΣ interface (unit1 (X)).

**[Syntax]**

```c
void R_DSMIF_X_Start ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
Filtering is stopped of $\Delta \Sigma$ interface (unit1 (X)).

**[Syntax]**

```c
void R_DSMIF_X_Stop ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
**r_dsmif_uvw_overcurrent_interrupt**

Performs processing in response to the overcurrent interrupt of ΔΣ interface (unit0 (U, V, W)).

**[Syntax]**

```c
void    r_dsmif_uvw_overcurrent_interrupt ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
r_dsmif_uvw_totalcurrent_interrupt

Performs processing in response to the total current interrupt of $\Delta \Sigma$ interface (unit 0 (U, V, W)).

[Syntax]

```c
void r_dsmif_uvw_totalcurrent_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.
r_dsmif_uvw_shortcircuit_interrupt

Performs processing in response to the shortcircuit interrupt of ΔΣ interface (unit0 (U, V, W)).

[Syntax]

```c
void r_dsmif_uvw_shortcircuit_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.
**r_dsmif_x_overcurrent_interrupt**

Performs processing in response to the overcurrent interrupt of ΔΣ interface (unit1 (X)).

**[Syntax]**

```c
void r_dsmif_x_overcurrent_interrupt ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
r_dsmif_x_shortcircuit_interrupt

Performs processing in response to the short circuit interrupt of ΔΣ interface (unit1 (X)).

**[Syntax]**

```c
void r_dsmif_x_shortcircuit_interrupt ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.23 Memory Protection Unit

Below is a list of API functions output by the Code Generator for Memory Protection Unit use

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_MPU_Create</td>
<td>Performs initialization required to control the Memory Protection Unit.</td>
</tr>
</tbody>
</table>
R_MPU_Create

Performs initialization required to control the Memory Protection Unit.

[Syntax]

```c
void R_MPU_Create ( void );
```

[Argument(s)]

None.

[Return value]

None.
### 3.2.24 Error Control Module

Below is a list of API functions output by the Code Generator for Error Control Module use.

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_ECM_Create</td>
<td>Performs initialization required to control the error control module.</td>
</tr>
<tr>
<td>R_ECM_Create_UserInit</td>
<td>Performs user-defined initialization relating to the error control module.</td>
</tr>
<tr>
<td>r_ecm_nmi_interrupt</td>
<td>Performs processing in response to the ECM non-maskable interrupt.</td>
</tr>
<tr>
<td>r_ecm_compareerror_interrupt</td>
<td>Performs processing in response to the ECM compare error interrupt.</td>
</tr>
<tr>
<td>R_ECM_Pseudo_WDT0_Error_Start</td>
<td>Generates corresponding pseudo error of WDT overflow / refresh error (error source 1) (Cortex-R4F).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_WDT0_Error_Stop</td>
<td>Pseudo error not generated WDT overflow / refresh error (error source 1) (Cortex-R4F).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_IWDTa_Error_Start</td>
<td>Generates corresponding pseudo error of IWDTa overflow / refresh error (error source 3).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_IWDTa_Error_Stop</td>
<td>Pseudo error not generated IWDTa overflow / refresh error (error source 3).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_CGC_Error_Start</td>
<td>Enables issuing the pseudo error “Main clock oscillation stop detection (error source 20)”.</td>
</tr>
<tr>
<td>R_ECM_Pseudo_CGC_Error_Stop</td>
<td>Disables issuing the pseudo error “Main clock oscillation stop detection (error source 20)”.</td>
</tr>
<tr>
<td>R_ECM_Pseudo_ADC_Unit0_Error_Start</td>
<td>Generates corresponding pseudo error of 12-bit A/D converter unit0 overwrite interrupt (error source 24).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_ADC_Unit0_Error_Stop</td>
<td>Pseudo error not generated 12-bit A/D converter unit0 overwrite interrupt (error source 24).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Start</td>
<td>Generates corresponding pseudo error of 12-bit A/D converter unit1 overwrite interrupt (error source 25).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_ADC_Unit1_Error_Stop</td>
<td>Pseudo error not generated 12-bit A/D converter unit1 overwrite interrupt (error source 25).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_DSMIF_UVWovercurrent_Error_Start</td>
<td>Generates corresponding pseudo error of ΔΣ interface UVW overcurrent abnormality detection error (error source 26).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_DSMIF_UVWovercurrent_Error_Stop</td>
<td>Pseudo error not generated ΔΣ interface UVW overcurrent abnormality detection error (error source 26).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_DSMIF_UVWtotalcurrent_Error_Start</td>
<td>Generates corresponding pseudo error of ΔΣ interface UVW total current abnormality detection error (error source 27).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_DSMIF_UVWtotalcurrent_Error_Stop</td>
<td>Pseudo error not generated ΔΣ interface UVW total current abnormality detection error (error source 27).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_DSMIF_UVWshortcircuit_Error_Start</td>
<td>Generates corresponding pseudo error of ΔΣ interface UVW short circuit abnormality detection error (error source 28).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_DSMIF_UVWshortcircuit_Error_Stop</td>
<td>Pseudo error not generated ΔΣ interface UVW short circuit abnormality detection error (error source 28).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_DSMIF_Xovercurrent_Error_Start</td>
<td>Generates corresponding pseudo error of ΔΣ interface X overcurrent abnormality detection error (error source 29).</td>
</tr>
</tbody>
</table>
### API Function Name

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_ECM_Pseudo_DSMIF_Xovercurrent_Error_Stop</td>
<td>Pseudo error not generated ΔΣ interface X overcurrent abnormality detection error (error source 29).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_DSMIF_Xshortcircuit_Error_Start</td>
<td>Generates corresponding pseudo error of ΔΣ interface X short circuit abnormality detection error (error source 31).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_DSMIF_Xshortcircuit_Error_Stop</td>
<td>Pseudo error not generated ΔΣ interface X short circuit abnormality detection error (error source 31).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_DOC_Error_Start</td>
<td>Generates corresponding pseudo error of DOC operation error (error source 32).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_DOC_Error_Stop</td>
<td>Pseudo error not generated DOC operation error (error source 32).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_BSC_Error_Start</td>
<td>Generates corresponding pseudo error of bus state controller detecting a long access wait state caused by the external WAIT pin (error source 34).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_BSC_Error_Stop</td>
<td>Pseudo error not generated bus state controller detecting a long access wait state caused by the external WAIT pin (error source 34).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_Error35_Error_Start</td>
<td>Generates corresponding pseudo error of extended pseudo error 35 (error source 35).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_Error35_Error_Stop</td>
<td>Pseudo error not generated extended pseudo error 35 (error source 35).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_Error36_Error_Start</td>
<td>Generates corresponding pseudo error of extended pseudo error 36 (error source 36).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_Error36_Error_Stop</td>
<td>Pseudo error not generated extended pseudo error 36 (error source 36).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_Error37_Error_Start</td>
<td>Generates corresponding pseudo error of extended pseudo error 37 (error source 37).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_Error37_Error_Stop</td>
<td>Pseudo error not generated extended pseudo error 37 (error source 37).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_Error38_Error_Start</td>
<td>Generates corresponding pseudo error of extended pseudo error 38 (error source 38).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_Error38_Error_Stop</td>
<td>Pseudo error not generated extended pseudo error 38 (error source 38).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_Error39_Error_Start</td>
<td>Generates corresponding pseudo error of extended pseudo error 39 (error source 39).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_Error39_Error_Stop</td>
<td>Pseudo error not generated extended pseudo error 39 (error source 39).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_Error40_Error_Start</td>
<td>Generates corresponding pseudo error of extended pseudo error 40 (error source 40).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_Error40_Error_Stop</td>
<td>Pseudo error not generated extended pseudo error 40 (error source 40).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_Error41_Error_Start</td>
<td>Generates corresponding pseudo error of extended pseudo error 41 (error source 41).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_Error41_Error_Stop</td>
<td>Pseudo error not generated extended pseudo error 41 (error source 41).</td>
</tr>
<tr>
<td>R_ECM_Pseudo_ECM_CompareError_Error_Start</td>
<td>Generates corresponding pseudo error of ECM compare error (error source 93).</td>
</tr>
</tbody>
</table>
### API Function Name

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>R_ECM_Pseudo_ECM_CompareError_Error_Stop</code></td>
<td>Pseudo error not generated ECM compare error (error source 93).</td>
</tr>
<tr>
<td><code>R_ECM_Pseudo_ECM_DelayTimerOverflow_Error_Start</code></td>
<td>Generates corresponding pseudo error of ECM delay timer overflow (error source 94).</td>
</tr>
<tr>
<td><code>R_ECM_Pseudo_ECM_DelayTimerOverflow_Error_Stop</code></td>
<td>Pseudo error not generated ECM delay timer overflow (error source 94).</td>
</tr>
</tbody>
</table>
**R_ECM_Create**

Performs initialization required to control the error control module.

**[Syntax]**

```c
void R_ECM_Create ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_ECM_Create_UserInit

Performs user-defined initialization relating to the error control module.
Remark This API function is called as the R_ECM_Create callback routine.

[Syntax]

```c
void R_ECM_Create_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.
r_ecm_nmi_interrupt

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

[Syntax]

```c
void r_ecm_nmi_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.
r_ecm_errd_interrupt

Performs processing in response to the error detection.

[Syntax]

```
void r_ecm_errd_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.
r_ecm_compareerror_interrupt

Performs processing in response to the ECM compare error interrupt.

[Syntax]

```c
void r_ecm_compareerror_interrupt(void);
```

[Argument(s)]

None.

[Return value]

None.
R_ECM_Pseudo_WDT0_Error_Start

Generates corresponding pseudo error of WDT overflow / refresh error (error source 1) (Cortex-R4F).

[Syntax]

```c
void R_ECM_Pseudo_WDT0_Error_Start ( void );
```

[Argument(s)]
None.

[Return value]
None.
**R_ECM_Pseudo_WDT0_Error_Stop**

Pseudo error not generated WDT overflow / refresh error (error source 1) (Cortex-R4F).

**[Syntax]**

```c
void R_ECM_Pseudo_WDT0_Error_Stop ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
### R_ECM_Pseudo_IWDTa_Error_Start

Generates corresponding pseudo error of IWDTa overflow / refresh error (error source 3).

**[Syntax]**

```c
void R_ECM_Pseudo_IWDTa_Error_Start ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_ECM_Pseudo_IWDTa_Error_Stop

Pseudo error not generated IWDTa overflow / refresh error (error source 3).

[Syntax]

```c
void R_ECM_Pseudo_IWDTa_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.
R_ECM_Pseudo_CGC_Error_Start

Enables issuing the pseudo error “Main clock oscillation stop detection (error source 20”).

[Syntax]

```
void R_ECM_Pseudo_CGC_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.
Disables issuing the pseudo error "Main clock oscillation stop detection (error source 20)".

[Syntax]

```c
void R_ECM_Pseudo_CGC_Error_Stop ( void );
```

[Argument(s)]
None.

[Return value]
None.
Generates corresponding pseudo error of 12-bit A/D converter unit0 overwrite interrupt (error source 24).

**[Syntax]**

```c
void R_ECM_Pseudo_ADC_Unit0_Error_Start ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_ECM_Pseudo_ADC_Unit0_Error_Stop

Pseudo error not generated 12-bit A/D converter unit0 overwrite interrupt (error source 24).

[Syntax]

```c
void R_ECM_Pseudo_ADC_Unit0_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.
Generates corresponding pseudo error of 12-bit A/D converter unit1 overwrite interrupt (error source 25).

[Syntax]

```c
void R_ECM_Pseudo_ADC_Unit1_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.
Pseudo error not generated 12-bit A/D converter unit1 overwrite interrupt (error source 25).

**[Syntax]**

```c
void R_ECM_Pseudo_ADC_Unit1_Error_Stop ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
Generates corresponding pseudo error of ΔΣ interface UVW overcurrent abnormality detection error (error source 26).

**Syntax**

```c
void R_ECM_Pseudo_DSMIF_UVWovercurrent_Error_Start ( void );
```

**Argument(s)**

None.

**Return value**

None.
R_ECM_Pseudo_DSMIF_UVWovercurrent_Error_Stop

Pseudo error not generated ΔΣ interface UVW overcurrent abnormality detection error (error source 26).

[Syntax]

```c
void R_ECM_Pseudo_DSMIF_UVWovercurrent_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.
Generates corresponding pseudo error of $\Delta \Sigma$ interface UVW total current abnormality detection error (error source 27).

**[Syntax]**

```c
void R_ECM_Pseudo_DSIFMIF_UVWtotalcurrent_Error_Start ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
### R_ECM_Pseudo_DSMIF_UVWtotalcurrent_Error_Stop

Pseudo error not generated ΔΣ interface UVW total current abnormality detection error (error source 27).

**[Syntax]**

```c
void R_ECM_Pseudo_DSMIF_UVWtotalcurrent_Error_Stop ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
Generates corresponding pseudo error of $\Delta \Sigma$ interface UVW short circuit abnormality detection error (error source 28).

**Syntax**

```c
void R_ECM_Pseudo_DSMIF_UVWshortcircuit_Error_Start ( void );
```

**Argument(s)**

None.

**Return value**

None.
Pseudo error not generated ΔΣ interface UVW short circuit abnormality detection error (error source 28).

**[Syntax]**

```c
void R_ECM_Pseudo_DSMIF_UVWshortcircuit_Error_Stop ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_ECM_Pseudo_DSMIF_Xovercurrent_Error_Start

Generates corresponding pseudo error of ΔΣ interface X overcurrent abnormality detection error (error source 29).

[Syntax]

```c
void R_ECM_Pseudo_DSMIF_Xovercurrent_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.
**R_ECM_Pseudo_DSMIF_Xovercurrent_Error_Stop**

Pseudo error not generated ΔΣ interface X overcurrent abnormality detection error (error source 29).

**[Syntax]**

```c
void R_ECM_Pseudo_DSMIF_Xovercurrent_Error_Stop ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
### R_ECM_Pseudo_DSMIF_Xshortcircuit_Error_Start

Generates corresponding pseudo error of $\Delta\Sigma$ interface X short circuit abnormality detection error (error source 31).

**[Syntax]**

```c
void R_ECM_Pseudo_DSMIF_Xshortcircuit_Error_Start ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_ECM_Pseudo_DSMIF_Xshortcircuit_Error_Stop

Pseudo error not generated ΔΣ interface X short circuit abnormality detection error (error source 31).

[Syntax]

```c
void R_ECM_Pseudo_DSMIF_Xshortcircuit_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.
R_ECM_Pseudo_DOC_Error_Start

Generates corresponding pseudo error of DOC operation error (error source 32).

[Syntax]

```c
void R_ECM_Pseudo_DOC_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.
R_ECM_Pseudo_DOC_Error_Stop

Pseudo error not generated DOC operation error (error source 32).

[Syntax]

\[
\text{void R_ECM_Pseudo_DOC_Error_Stop ( void );}
\]

[Argument(s)]

None.

[Return value]

None.
R_ECM_Pseudo_BSC_Error_Start

Generates corresponding pseudo error of bus state controller detecting a long access wait state caused by the external WAIT pin (error source 34).

[Syntax]

```c
void R_ECM_Pseudo_BSC_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.
Pseudo error not generated bus state controller detecting a long access wait state caused by the external WAIT pin (error source 34).

**[Syntax]**

```c
void R_ECM_Pseudo_BSC_Error_Stop ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
Generates corresponding pseudo error of extended pseudo error 35 (error source 35).

**[Syntax]**

```c
void    R_ECM_Pseudo_Error35_Error_Start ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_ECM_Pseudo_Error35_Error_Stop

Pseudo error not generated extended pseudo error 35 (error source 35).

[Syntax]

```c
void R_ECM_Pseudo_Error35_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.
Generates corresponding pseudo error of extended pseudo error 36 (error source 36).

**[Syntax]**

```c
void R_ECM_Pseudo_Error36_Error_Start ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
### R_ECM_Pseudo_Error36_Error_Stop

Pseudo error not generated extended pseudo error 36 (error source 36).

**[Syntax]**

```c
void R_ECM_Pseudo_Error36_Error_Stop ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
Generates corresponding pseudo error of extended pseudo error 37 (error source 37).

**[Syntax]**

```c
void    R_ECM_Pseudo_Error37_Error_Start ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_ECM_Pseudo_Error37_Error_Stop

Pseudo error not generated extended pseudo error 37 (error source 37).

[Syntax]

```c
void R_ECM_Pseudo_Error37_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.
R_ECM_Pseudo_Error38_Error_Start

Generates corresponding pseudo error of extended pseudo error 38 (error source 38).

[Syntax]

```c
void R_ECM_Pseudo_Error38_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.
**R_ECM_Pseudo_Error38_Error_Stop**

Pseudo error not generated extended pseudo error 38 (error source 38).

**[Syntax]**

```c
void R_ECM_Pseudo_Error38_Error_Stop ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
Generates corresponding pseudo error of extended pseudo error 39 (error source 39).

**[Syntax]**

```c
void    R_ECM_Pseudo_Error39_Error_Start ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_ECM_Pseudo_Error39_Error_Stop

Pseudo error not generated extended pseudo error 39 (error source 39).

[Syntax]

```c
void R_ECM_Pseudo_Error39_Error_Stop ( void );
```

[Argument(s)]
None.

[Return value]
None.
R_ECM_Pseudo_Error40_Error_Start

Generates corresponding pseudo error of extended pseudo error 40 (error source 40).

[Syntax]

```c
void R_ECM_Pseudo_Error40_Error_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.
### R_ECM_Pseudo_Error40_Error_Stop

Pseudo error not generated extended pseudo error 40 (error source 40).

**[Syntax]**

```c
void R_ECM_Pseudo_Error40_Error_Stop ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
Generates corresponding pseudo error of extended pseudo error 41 (error source 41).

**[Syntax]**

```c
void R_ECM_Pseudo_Error41_Error_Start ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_ECM_Pseudo_Error41_Error_Stop

Pseudo error not generated extended pseudo error 41 (error source 41).

[Syntax]

```c
void    R_ECM_Pseudo_Error41_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.
Generates corresponding pseudo error of ECM compare error (error source 93).

**[Syntax]**

```c
void R_ECM_Pseudo_ECM_CompareError_Error_Start ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_ECM_Pseudo_ECM_CompareError_Error_Stop

Pseudo error not generated ECM compare error (error source 93).

[Syntax]

```c
void R_ECM_Pseudo_ECM_CompareError_Error_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.
Generates corresponding pseudo error of ECM delay timer overflow (error source 94).

**Syntax**

```c
void R_ECM_Pseudo_ECM_DelayTimerOverflow_Error_Start ( void );
```

**Argument(s)**

None.

**Return value**

None.
Pseudo error not generated ECM delay timer overflow (error source 94).

**[Syntax]**

```c
void R_ECM_Pseudo_ECM_DelayTimerOverflow_Error_Stop ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
### 3.2.25 12-Bit A/D Converter

Below is a list of API functions output by the Code Generator for 12-Bit A/D Converter use.

Table 3.26  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 required 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_s12ad_s12adn_interrupt</td>
<td>Performs processing in response to the A/D conversion end interrupt.</td>
</tr>
<tr>
<td>r_s12ad_s12gbadin_interrupt</td>
<td>Performs processing in response to the group B A/D conversion end interrupt.</td>
</tr>
<tr>
<td>r_s12ad_s12aden_interrupt</td>
<td>Performs processing in response to the A/D error interrupt.</td>
</tr>
<tr>
<td>R_S12ADn_Start</td>
<td>Starts the A/D conversion.</td>
</tr>
<tr>
<td>R_S12ADn_Stop</td>
<td>Ends the 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_s12ad_s12cmpn_interrupt</td>
<td>Performs processing in response to the compare interrupt.</td>
</tr>
</tbody>
</table>
R_S12ADn_Create

Performs initialization required to control the 12-Bit A/D converter.

[Syntax]

```c
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_s12ad_s12adn_interrupt

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

[Syntax]

```c
void r_s12ad_s12adn_interrupt ( void );
```

Remark $n$ is the unit number.

[Argument(s)]

None.

[Return value]

None.
r_s12ad_s12gbadin_interrupt

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

[Syntax]

```c
static void r_s12ad_s12gbadin_interrupt ( void );
```

Remark

\( n \) is the unit number.

[Argument(s)]

None.

[Return value]

None.
r_s12ad_s12aden_interrupt

Performs processing in response to the A/D error interrupt.

[Syntax]

```c
void r_s12ad_s12aden_interrupt ( void );
```

Remark

\( n \) is the unit number.

[Argument(s)]

None.

[Return value]

None.
**R_S12ADn_Start**

Starts the A/D conversion.

**[Syntax]**

```c
void R_S12ADn_Start ( void );
```

Remark \( n \) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
Ends the 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 conversion result.

**[Syntax]**

```c
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:</td>
<td>Input channel AN000</td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL1:</td>
<td>Input channel AN001</td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL2:</td>
<td>Input channel AN002</td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL3:</td>
<td>Input channel AN003</td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL4:</td>
<td>Input channel AN004</td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL6:</td>
<td>Input channel AN006</td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL8:</td>
<td>Input channel AN008</td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL9:</td>
<td>Input channel AN009</td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL10:</td>
<td>Input channel AN010</td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL11:</td>
<td>Input channel AN011</td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL12:</td>
<td>Input channel AN012</td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL13:</td>
<td>Input channel AN013</td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL14:</td>
<td>Input channel AN014</td>
</tr>
<tr>
<td></td>
<td>ADCHANNEL15:</td>
<td>Input channel AN015</td>
</tr>
<tr>
<td></td>
<td>ADTEMPSENSOR:</td>
<td>Extended analog input (temperature sensor output)</td>
</tr>
<tr>
<td></td>
<td>ADINTERREFVOLT:</td>
<td>Extended analog input (internal reference voltage)</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_ComporeValue

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><code>uint16_t reg_value0</code></td>
<td>Register value set to the compare revel register 0</td>
</tr>
<tr>
<td>I</td>
<td><code>uint16_t reg_value1</code></td>
<td>Register value set to the compare revel register 1</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
r_s12ad_s12cmpn_interrupt

Performs processing in response to the compare interrupt.

[Syntax]

```
static void r_s12ad_s12cmpn_interrupt ( void );
```

Remark

\( n \) is the unit number.

[Argument(s)]

None.

[Return value]

None.
### 3.2.26 Data Operation Circuit

Below is a list of API functions output by the Code Generator for Data Operation Circuit use.

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_DOC_Create</td>
<td>Performs initialization required to control the data operation circuit.</td>
</tr>
<tr>
<td>R_DOC_Create_Init</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.

**[Syntax]**

```
void R_DOC_Create ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
Performs user-defined initialization relating to the data operation circuit.

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

**R_DOC_Create_UserInit**

```c
void R_DOC_Create_UserInit ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
r_doc_dopcfs_interrupt

Performs processing in response to the data operation circuit interrupt.

[Syntax]

```c
void r_doc_dopcfs_interrupt ( void );
```

[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.

Remarks 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).

Remarks 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
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></td>
<td>COMPARE_MISMATCH: Data comparison mode (mismatch)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COMPARE_MATCH: Data comparison mode (match)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ADDITION: Data addition mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBTRACTION: Data subtraction mode</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.
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 depend on the product’s quality grade, as indicated below.

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

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

Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; underwater 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.

(Rev.4.0-1 November 2017)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact information

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

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.
## Revision Record

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Page</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>Apr 01, 2015</td>
<td>-</td>
<td>First Edition issued</td>
</tr>
<tr>
<td>1.01</td>
<td>Dec 01, 2017</td>
<td>-</td>
<td>Correction of errors, Creation of the description for omissions</td>
</tr>
</tbody>
</table>
| 1.02  | Mar 13, 2020 | 23 - 24 | Added following functions.  
NESTED_INTERRUPT_PUSH  
NESTED_INTERRUPT_POP  |
|       |              | 172 - 179 | Deleted following functions.  
r_scifan_telifn_interrupt  
r_scifan_erifn_interrupt |
|       |              | 188, 198 | Added following functions.  
R_RIlCn_Master_Send_Without_Stop |
|       |              | 238, 245 - 249 | Added following functions.  
r_dsmif_uvw_overcurrent_interrupt  
r_dsmif_uvw_totalcurrent_interrupt  
r_dsmif_uvw_shortcircuit_interrupt  
r_dsmif_x_overcurrent_interrupt  
r_dsmif_x_shortcircuit_interrupt |
|       |              | 251 - 252 | Added following functions.  
R_MPU_Create |