Smart Configurator
User's Manual: RH850 API Reference

Target Device
RH850 Family

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

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

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

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

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

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

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

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

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

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

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.
How to Use This Manual

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

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

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

How to Read This Manual
It is assumed that the readers of this manual have general knowledge of electricity, logic circuits, and microcontrollers.

Conventions
Deata significance: Higher digits on the left and lower digits on the right
Active low representation: XXX (overscore over pin or signal name)
Note: Footnote for item marked with Note in the text
Caution: Information requiring particular attention
Remark: Supplementary information
Numeric representation: Decimal ... XXXX
Hexadecimal ... 0xXXXX

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

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

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

3. API FUNCTIONS ........................................................................................................................................... 17
   3.1 Overview .................................................................................................................................................. 17
   3.2 Function Reference .................................................................................................................................. 18
      3.2.1 Common ........................................................................................................................................... 20
      3.2.2 Group Scan Mode ADCA .................................................................................................................. 28
      3.2.3 CSI Slave .......................................................................................................................................... 39
      3.2.4 CSI Master .......................................................................................................................................... 52
      3.2.5 Interrupt ........................................................................................................................................... 65
      3.2.6 Input Interval Timer ........................................................................................................................... 72
      3.2.7 Input Pulse Interval Measurement .................................................................................................... 78
      3.2.8 Interval Timer ..................................................................................................................................... 85
      3.2.9 Triangle PWM Output ......................................................................................................................... 91
      3.2.10 OS Timer .......................................................................................................................................... 97
      3.2.11 Port .................................................................................................................................................. 104
      3.2.12 PWM Output .................................................................................................................................... 107
      3.2.13 Stand-by Controller .......................................................................................................................... 113
      3.2.14 UART Interface ................................................................................................................................. 125
      3.2.15 Watchdog Timer ................................................................................................................................. 138
      3.2.16 Clock Divide .................................................................................................................................... 143
      3.2.17 Data CRC ......................................................................................................................................... 150
      3.2.18 Delay Count ..................................................................................................................................... 158
      3.2.19 DMA Controller ............................................................................................................................... 164
      3.2.20 External Event Count ......................................................................................................................... 176
      3.2.21 Input Period Count Detection .......................................................................................................... 182
      3.2.22 Input Position Detection .................................................................................................................. 189
      3.2.23 Input Pulse Interval Judgment ......................................................................................................... 196
      3.2.24 Input Signal Width Judgment .......................................................................................................... 196
      3.2.25 Input Signal Width Measurement .................................................................................................... 202
      3.2.26 Key Interrupt Function .................................................................................................................... 208
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.27</td>
<td>One Pulse Output</td>
<td>221</td>
</tr>
<tr>
<td>3.2.28</td>
<td>One-shot Pulse Output</td>
<td>227</td>
</tr>
<tr>
<td>3.2.29</td>
<td>Overflow Interrupt Output (Input period count detecting)</td>
<td>233</td>
</tr>
<tr>
<td>3.2.30</td>
<td>Overflow Interrupt Output (Width measurement)</td>
<td>240</td>
</tr>
<tr>
<td>3.2.31</td>
<td>IIC Master Mode</td>
<td>247</td>
</tr>
<tr>
<td>3.2.32</td>
<td>IIC Slave Mode</td>
<td>264</td>
</tr>
<tr>
<td>3.2.33</td>
<td>Real-Time Clock</td>
<td>280</td>
</tr>
</tbody>
</table>

Revision Record .................................................................................................................. 305
1. GENERAL

This chapter gives an overview of the driver code generator (hereafter abbreviated as Code Generator) of the Smart Configurator.

1.1 Overview

His tool can output source code (device driver programs as C source and header files) for controlling peripheral modules (clock generation circuit, voltage detection circuit, etc.) of the device by using a GUI to set various types of information on the requirements of the project.

1.2 Features

The features of the Code Generator are as follows.

- Generating code
  The Code Generator outputs not only device driver files in accord with the information set in the GUI but also a complete set of programs for the build environment, such as a sample program containing the call of the main function.

- Reporting
  Information that was set by using the Code Generator can be output to files in various formats and used as design documentation.

- Renaming
  Default names are given to folders and files output by the Code Generator and to the API functions in the source code, but these can be changed to user-specified names.

- Protecting user code
  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 */

  Code written by the user between these comments will be preserved even when the code is generated again.
2. OUTPUT FILES

This chapter explains the file output by the Code Generator.

2.1 Description

The Code Generator outputs the following files.

Table 2.1 Output File List (1/10)

<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>R_MAIN_UserInit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_systeminit.c</td>
<td>R_Systeminit</td>
</tr>
<tr>
<td></td>
<td>r_cg_cg.c</td>
<td>R_CGC_Create</td>
</tr>
<tr>
<td></td>
<td>r_cg_cg_user.c</td>
<td>R_CGC_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td>r_cg_cg.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_intvector.c</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_macrodriver.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_userdefine.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_smc_interrupt.c</td>
<td>R_Interrupt_Create</td>
</tr>
<tr>
<td></td>
<td>r_smc_interrupt.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pin.c</td>
<td>R_Pins_Create</td>
</tr>
<tr>
<td></td>
<td>Pin.h</td>
<td></td>
</tr>
</tbody>
</table>
Table 2.2 Output File List (2/10)

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>File Name</th>
<th>API Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Group Scan Mode</strong></td>
<td><strong>ADCA</strong></td>
<td><strong>Config_ADCA_n.c</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_ADCA_n_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_ADCA_n_Halt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_ADCA_n_ScanGroupx_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_ADCA_n_ScanGroupx_OperationOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_ADCA_n_ScanGroupx_GetResult</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_ADCA_n_TH_Groupk_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_ADCA_n_TH_Sampling_Start</td>
</tr>
<tr>
<td></td>
<td><strong>user.c</strong></td>
<td>R_Config_ADCA_n_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_ADCA_n_error_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_ADCA_n_scan_groupx_end_interrupt</td>
</tr>
<tr>
<td></td>
<td><strong>h</strong></td>
<td></td>
</tr>
<tr>
<td><strong>CSI Slave</strong></td>
<td><strong>Config_CSIGN_n.c</strong></td>
<td>R_Config_CSIGN_n_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_CSIGN_n_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_CSIGN_n_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_CSIGN_n_Send</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_CSIGN_n_Receive</td>
</tr>
<tr>
<td></td>
<td><strong>user.c</strong></td>
<td>R_Config_CSIGN_n_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_CSIGN_n_interrupt_send</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_CSIGN_n_interrupt_receive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_CSIGN_n_interrupt_error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_CSIGN_n_callback_sendend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_CSIGN_n_callback_receiveend</td>
</tr>
<tr>
<td></td>
<td><strong>h</strong></td>
<td></td>
</tr>
<tr>
<td><strong>CSI Master</strong></td>
<td><strong>Config_CSIGN_n.c</strong></td>
<td>R_Config_CSIGN_n_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_CSIGN_n_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_CSIGN_n_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_CSIGN_n_Send</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_CSIGN_n_Receive</td>
</tr>
<tr>
<td></td>
<td><strong>user.c</strong></td>
<td>R_Config_CSIGN_n_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_CSIGN_n_interrupt_send</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_CSIGN_n_interrupt_receive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_CSIGN_n_interrupt_error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_CSIGN_n_callback_sendend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_CSIGN_n_callback_receiveend</td>
</tr>
<tr>
<td></td>
<td><strong>h</strong></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>Interrupt</td>
<td>Config_ICU.c</td>
<td>R_Config_ICU_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_ICU_INTPn_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_ICU_INTPn_Stop</td>
</tr>
<tr>
<td></td>
<td>Config_ICU_user.c</td>
<td>R_Config_ICU_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_ICU_INTPn_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_ICU_nmi_interrupt</td>
</tr>
<tr>
<td></td>
<td>Config_ICU.h</td>
<td></td>
</tr>
<tr>
<td>Input Interval Timer</td>
<td>Config_TAUBn_m.c</td>
<td>R_Config_TAUBn_m_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Stop</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_m_user.c</td>
<td>R_Config_TAUBn_m_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_TAUBn_m_interrupt</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_m.h</td>
<td></td>
</tr>
<tr>
<td>Input Pulse Interval Measurement</td>
<td>Config_TAUBn_m.c</td>
<td>R_Config_TAUBn_m_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Get_PulseWidth</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_m_user.c</td>
<td>R_Config_TAUBn_m_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_TAUBn_m_interrupt</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_m.h</td>
<td></td>
</tr>
<tr>
<td>Interval Timer</td>
<td>Config_TAUBn_m.c</td>
<td>R_Config_TAUBn_m_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m.Stop</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_m_user.c</td>
<td>R_Config_TAUBn_m_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_TAUBn_m_interrupt</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_m.h</td>
<td></td>
</tr>
<tr>
<td>Triangle PWM Output</td>
<td>Config_TAUBn.c</td>
<td>R_Config_TAUBn_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_Stop</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_user.c</td>
<td>R_Config_TAUBn_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_TAUBn_channelm_interrupt</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn.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>OS Timer</td>
<td>Config_OSTMn.c</td>
<td>R_Config_OSTMn_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_OSTMn_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_OSTMn_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_OSTMn_Set_CompareValue</td>
</tr>
<tr>
<td></td>
<td>Config_OSTMn_user.c</td>
<td>R_Config_OSTMn_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_OSTMn_interrupt</td>
</tr>
<tr>
<td></td>
<td>Config_OSTMn.h</td>
<td>--</td>
</tr>
<tr>
<td>Port</td>
<td>Config_PORT.c</td>
<td>R_Config_PORT_Create</td>
</tr>
<tr>
<td></td>
<td>Config_PORT_user.c</td>
<td>R_Config_PORT_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td>Config_PORT.h</td>
<td>--</td>
</tr>
<tr>
<td>PWM Output</td>
<td>Config_TAUBn.c</td>
<td>R_Config_TAUBn_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_Stop</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_user.c</td>
<td>R_Config_TAUBn_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_TAUBn_channelm_interrupt</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn.h</td>
<td>--</td>
</tr>
<tr>
<td>Stand-by Controller</td>
<td>Config_STBC.c</td>
<td>R_Config_STBC_Prepare_Stop_Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_STBC_Start_Stop_Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_STBC_Prepare_Deep_Stop_Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_STBC_Start_Deep_Stop_Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_STBC_Deep_Stop_Loop</td>
</tr>
<tr>
<td></td>
<td>Config_STBC_user.c</td>
<td>R_Config_STBC_Prepare_Stop_Mode_Set_Peripheral</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_STBC_Prepare_Stop_Mode_Set_Interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_STBC_Prepare_Stop_Mode_Set_Clock_Mask</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_STBC_Prepare_Stop_Mode_Set_Clock_Source</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_STBC_Prepare_Deep_Stop_Mode_Set_Peripheral</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_STBC_Prepare_Deep_Stop_Mode_Set_Interrupt</td>
</tr>
<tr>
<td></td>
<td>Config_STBC.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>UART Interface</td>
<td>Config_UARTn.c</td>
<td>R_Config_UARTn_Create R_Config_UARTn_Start R_Config_UARTn_Stop R_Config_UARTn_Send R_Config_UARTn_Receive</td>
</tr>
<tr>
<td></td>
<td>Config_UARTn_user.c</td>
<td>R_Config_UARTn_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_UARTn_interrupt_send</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_UARTn_interrupt_receive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_UARTn_interrupt_error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_UARTn_callback_sendend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_UARTn_callback_receiveend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_UARTn_callback_error</td>
</tr>
<tr>
<td></td>
<td>Config_UARTn.h</td>
<td>—</td>
</tr>
<tr>
<td>Watchdog Timer</td>
<td>Config_WDTn.c</td>
<td>R_Config_WDTn_Create R_Config_WDTn_Restart</td>
</tr>
<tr>
<td></td>
<td>Config_WDTn_user.c</td>
<td>R_Config_WDTn_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_WDTn_interrupt</td>
</tr>
<tr>
<td></td>
<td>Config_WDTn.h</td>
<td>—</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>File Name</th>
<th>API Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock Divide</td>
<td>Config_TAUBn.c</td>
<td>R_Config_TAUBn_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_Channelm_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_Channelm_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_Channelm_Get_PulseWidth</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_user.c</td>
<td>R_Config_TAUBn_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_TAUBn_channelsm_interrupt</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn.h</td>
<td>—</td>
</tr>
<tr>
<td>Data CRC</td>
<td>Config_DCRAn.c</td>
<td>R_Config_DCRAn_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_DCRAn_Input32bitData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_DCRAn_Input16bitData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_DCRAn_Input8bitData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_DCRAn_GetResult_32bitData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_DCRAn_GetResult_16bitData</td>
</tr>
<tr>
<td></td>
<td>Config_DCRAn_user.c</td>
<td>R_Config_DCRAn_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Config_DCRAn.h</td>
<td>—</td>
</tr>
<tr>
<td>Delay Count</td>
<td>Config_TAUDn_m.c</td>
<td>R_Config_TAUDn_m_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUDn_m_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUDn_m_Stop</td>
</tr>
<tr>
<td></td>
<td>Config_TAUDn_m_user.c</td>
<td>R_Config_TAUDn_m_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_TAUDn_m_interrupt</td>
</tr>
<tr>
<td></td>
<td>Config_TAUDn_m.h</td>
<td>—</td>
</tr>
<tr>
<td>DMA Controller</td>
<td>Config_DMACn.c</td>
<td>R_Config_DMACn_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DMACnm_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_DMACnm_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_DMACnm_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_DMACnm_Set_SoftwareTrigger</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_DMACnm_Suspend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_DMACnm_Resume</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_DMACnm_Suspend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_DMACnm_Resume</td>
</tr>
<tr>
<td></td>
<td>Config_DMACn_user.c</td>
<td>R_Config_DMACnm_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_DMACnm_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_DMACnm_callback_transfer_completion</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_DMACnm_callback_transfer_count_match</td>
</tr>
<tr>
<td></td>
<td>Config_DMACn.h</td>
<td>—</td>
</tr>
</tbody>
</table>
### Table 2.7 Output File List (7/10)

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>File Name</th>
<th>API Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>External Event Count</strong></td>
<td>Config_TAUBn_m.c</td>
<td>R_Config_TAUBn_m_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Stop</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_m_user.c</td>
<td>R_Config_TAUBn_m_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_TAUBn_m_interrupt</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_m.h</td>
<td>—</td>
</tr>
<tr>
<td><strong>Input Period Count Detection</strong></td>
<td>Config_TAUBn_m.c</td>
<td>R_Config_TAUBn_m_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Get_PulseWidth</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_m_user.c</td>
<td>R_Config_TAUBn_m_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_TAUBn_m_interrupt</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_m.h</td>
<td>—</td>
</tr>
<tr>
<td><strong>Input Position Detection Function</strong></td>
<td>Config_TAUBn_m.c</td>
<td>R_Config_TAUBn_m_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Get_PulseWidth</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_m_user.c</td>
<td>R_Config_TAUBn_m_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_TAUBn_m_interrupt</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_m.h</td>
<td>—</td>
</tr>
<tr>
<td><strong>Input Pulse Interval Judgment</strong></td>
<td>Config_TAUBn_m.c</td>
<td>R_Config_TAUBn_m_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Stop</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_m_user.c</td>
<td>R_Config_TAUBn_m_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_TAUBn_m_interrupt</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_m.h</td>
<td>—</td>
</tr>
<tr>
<td><strong>Input Signal Width Judgment</strong></td>
<td>Config_TAUBn_m.c</td>
<td>R_Config_TAUBn_m_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Stop</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_m_user.c</td>
<td>R_Config_TAUBn_m_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_TAUBn_m_interrupt</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_m.h</td>
<td>—</td>
</tr>
<tr>
<td><strong>Input Signal Width Measurement</strong></td>
<td>Config_TAUBn_m.c</td>
<td>R_Config_TAUBn_m_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Get_PulseWidth</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_m_user.c</td>
<td>R_Config_TAUBn_m_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_TAUBn_m_interrupt</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_m.h</td>
<td>—</td>
</tr>
</tbody>
</table>
### Table 2.8 Output File List (8/10)

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>File Name</th>
<th>API Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key Interrupt Function</strong></td>
<td>Config_KEY.c</td>
<td>R_Config_KEY_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_KEY_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_KEY_Stop</td>
</tr>
<tr>
<td></td>
<td>Config_KEY_user.c</td>
<td>R_Config_KEY_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_KEY_interrupt</td>
</tr>
<tr>
<td></td>
<td>Config_KEY.h</td>
<td></td>
</tr>
<tr>
<td><strong>One Pulse Output</strong></td>
<td>Config_TAUBn_m.c</td>
<td>R_Config_TAUBn_m_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Stop</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_m_user.c</td>
<td>R_Config_TAUBn_m_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_TAUBn_m_interrupt</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_m.h</td>
<td></td>
</tr>
<tr>
<td><strong>One-Shot Pulse Output</strong></td>
<td>Config_TAUBn_m.c</td>
<td>R_Config_TAUBn_m_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Stop</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_m_user.c</td>
<td>R_Config_TAUBn_m_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_TAUBn_m_interrupt</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_m.h</td>
<td></td>
</tr>
<tr>
<td><strong>Overflow Interrupt Output</strong></td>
<td>Config_TAUBn_m.c</td>
<td>R_Config_TAUBn_m_Create</td>
</tr>
<tr>
<td>(Input period count detecting)</td>
<td></td>
<td>R_Config_TAUBn_m_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Get_PulseWidth</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_m_user.c</td>
<td>R_Config_TAUBn_m_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_TAUBn_m_interrupt</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_m.h</td>
<td></td>
</tr>
<tr>
<td><strong>Overflow Interrupt Output</strong></td>
<td>Config_TAUBn_m.c</td>
<td>R_Config_TAUBn_m_Create</td>
</tr>
<tr>
<td>(Width Measurement)</td>
<td></td>
<td>R_Config_TAUBn_m_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_TAUBn_m_Get_PulseWidth</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_m_user.c</td>
<td>R_Config_TAUBn_m_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_TAUBn_m_interrupt</td>
</tr>
<tr>
<td></td>
<td>Config_TAUBn_m.h</td>
<td></td>
</tr>
</tbody>
</table>
Table 2.9 Output File List (9/10)

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>File Name</th>
<th>API Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IIC Master Mode</strong></td>
<td>Config_RIIcn.c</td>
<td>R_Config_RIIcn_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_RIIcn_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_RIIcn_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_RIIcn_Master_Send</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_RIIcn_Master_Receive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_RIIcn_StartCondition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_RIIcn_StopCondition</td>
</tr>
<tr>
<td></td>
<td>Config_RIIcn_user.c</td>
<td>R_Config_RIIcn_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_RIIcn_error_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_RIIcn_receive_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_RIIcn_transmit_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_RIIcn_transmitend_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_RIIcn_callback_transmitend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_RIIcn_callback_receiveend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_RIIcn_callback_receiveerror</td>
</tr>
<tr>
<td></td>
<td>Config_RIIcn.h</td>
<td>—</td>
</tr>
<tr>
<td><strong>IIC Slave Mode</strong></td>
<td>Config_RIIcn.c</td>
<td>R_Config_RIIcn_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_RIIcn_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_RIIcn_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_RIIcn_Slave_Send</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_RIIcn_Slave_Receive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_RIIcn_StartCondition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_Config_RIIcn_StopCondition</td>
</tr>
<tr>
<td></td>
<td>Config_RIIcn_user.c</td>
<td>R_Config_RIIcn_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_RIIcn_error_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_RIIcn_receive_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_RIIcn_transmit_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_RIIcn_transmitend_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_RIIcn_callback_transmitend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_RIIcn_callback_receiveend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_Config_RIIcn_callback_receiveerror</td>
</tr>
<tr>
<td></td>
<td>Config_RIIcn.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>Real-Time Clock</td>
<td>Config_RTC.c</td>
<td>R_Config_RTC_Create&lt;br&gt;R_Config_RTC_Start&lt;br&gt;R_Config_RTC_Stop&lt;br&gt;R_Config_RTC_Set_HourSystem&lt;br&gt;R_Config_RTC_Get_CounterBufferValue&lt;br&gt;R_Config_RTC_Get_CounterDirectValue&lt;br&gt;R_Config_RTC_Set_CounterValue&lt;br&gt;R_Config_RTC_Get_SubCounterValue&lt;br&gt;R_Config_RTC_Set_ErrorCorrectionValue&lt;br&gt;R_Config_RTC_Set_SubCounterCompareValue&lt;br&gt;R_Config_RTC_Get_AlarmValue&lt;br&gt;R_Config_RTC_Set_AlarmValue&lt;br&gt;R_Config_RTC_Set_AlarmOn&lt;br&gt;R_Config_RTC_Set_AlarmOff&lt;br&gt;R_Config_RTC_Set_ConstPeriodInterruptOn&lt;br&gt;R_Config_RTC_Set_ConstPeriodInterruptOff&lt;br&gt;R_Config_RTC_Set_1secondInterruptOn&lt;br&gt;R_Config_RTC_Set_1secondInterruptOff&lt;br&gt;R_Config_RTC_Set_RTCA1HZOn&lt;br&gt;R_Config_RTC_Set_RTCA1HZOff</td>
</tr>
<tr>
<td></td>
<td>Config_RTC_user.c</td>
<td>R_Config_RTC_Create_UserInit&lt;br&gt;r_Config_RTC_interrupt_alarm&lt;br&gt;r_Config_RTC_interrupt_periodic&lt;br&gt;r_Config_RTC_interrupt_1second</td>
</tr>
<tr>
<td></td>
<td>Config_RTC.h</td>
<td>--</td>
</tr>
</tbody>
</table>
This chapter describes the API functions output that are output by the Code Generator.

3.1 Overview

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

- **Macro names**
  These are in all-capital letters.
  Note that if a name includes a number as a prefix, the relevant number is equal to the hexadecimal value of the macro.

- **Local variable names**
  These are in low-case letters only.

- **Global variable names**
  These are prefixed with "g", and only the first letters of words that are elements of the names are capitals.

- **Names of pointers to global variables**
  These are prefixed with "gp", and only the first letters of words that are elements of the names are capitals.

- **Names of elements in enumeration specifiers “enum”**
  These are in all-capital letters.

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

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

Figure 3.1 Notation Format of API Functions

- **(1)** Name
  - Indicates the name of the API function.
- **(2)** Outline
  - Outlines the functions of the API function.
- **(3)** [Syntax]
  - Indicates the format to be used when describing an API function to be called in C language.
- **(4)** [Argument(s)]
  - API function arguments are explained in the following format.

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

  - **(a)** I/O
    - Argument classification
      - I ... Input argument
      - O ... Output argument
  - **(b)** Argument
    - Argument data type
  - **(c)** Description
    - Description of argument
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>Macro of return value</td>
</tr>
<tr>
<td>(b)</td>
<td>Description of return value</td>
</tr>
</tbody>
</table>

(5) [Return value]
3.2.1 Common

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

Table 3.1 API Functions: [Common]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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_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_Interrupt_Create</td>
<td>(Reserved function)</td>
</tr>
<tr>
<td>R_Pins_Create</td>
<td>This is a reference for setting the pin function.</td>
</tr>
</tbody>
</table>
This is a main function.

**[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.

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

[Syntax]

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

[Argument(s)]
None.

[Return value]
None.
**R_CGC_Create**

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

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

**[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.
RInterrupt_Create

Reserved function (Processing code is not output).

[Syntax]
void R_Interrupt_Create ( void );

[Argument(s)]
None.

[Return value]
None.
R_Pins_Create

This is a reference for setting the pin function (Other than I/O port).
For the pin function set on the [Pins] tab of the Smart Configurator, the setting code is output as a reference. Because it is a reference, it is not called from any function.

[Syntax]

```c
void     R_Pins_Create ( void );
```

[Argument(s)]
None.

[Return value]
None.
3.2.2 Group Scan Mode ADCA

Below is a list of API functions output by the Code Generator for group scan mode ADCA use.

Table 3.2 API Functions: [Group Scan Mode ADCA]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_Config_ADCAn_Create</td>
<td>Performs initialization necessary to control the group scan mode ADCA functions.</td>
</tr>
<tr>
<td>R_Config_ADCAn_Halt</td>
<td>Halts A/D converter.</td>
</tr>
<tr>
<td>R_Config_ADCAn_ScanGroupx_Start</td>
<td>Starts A/D converter of scan group.</td>
</tr>
<tr>
<td>R_Config_ADCAn_ScanGroupx_OperationOn</td>
<td>Starts scan group scan.</td>
</tr>
<tr>
<td>R_Config_ADCAn_ScanGroupx_GetResult</td>
<td>Reads the A/D conversion results of scan group.</td>
</tr>
<tr>
<td>R_Config_ADCAn_TH_Groupk_Start</td>
<td>Starts T&amp;H group hold.</td>
</tr>
<tr>
<td>R_Config_ADCAn_TH_Sampling_Start</td>
<td>Starts T&amp;H sampling.</td>
</tr>
<tr>
<td>R_Config_ADCAn_Create_UserInit</td>
<td>Performs user-defined initialization relating to the group scan mode ADCA functions.</td>
</tr>
<tr>
<td>r_Config_ADCAn_error_interrupt</td>
<td>Performs processing in response to the A/D error interrupt.</td>
</tr>
<tr>
<td>r_Config_ADCAn_scan_groupx_end_interrupt</td>
<td>Performs processing in response to the scan group end interrupt.</td>
</tr>
</tbody>
</table>
**R_Config_ADCan_Create**

Performs initialization necessary to control the group scan mode ADCA functions.

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

**[Syntax]**

```c
void R_Config_ADCan_Create ( void );
```

Remark \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_ADCA_n_Halt

Halts A/D converter.

[Syntax]

```c
void R_Config_ADCA_n_Halt ( void );
```

Remark  

$n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
### R_Config_ADCAn_ScanGroupx_Start

Starts A/D converter of scan group.

**[Syntax]**

```c
void R_Config_ADCAn_ScanGroupx_Start ( void );
```

**Remark**  
$n$ is the unit number, $x$ is the scan group number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_ADCan_ScanGroupx_OperationOn

Starts scan group scan.

[Syntax]

```c
void R_Config_ADCan_ScanGroupx_OperationOn ( void );
```

Remark  

\( n \) is the unit number, \( x \) is the scan group number.

[Argument(s)]

None.

[Return value]

None.
**R_Config_ADCAn_ScanGroupx_GetResult**

Reads the A/D conversion results of scan group.

### Syntax

```c
void R_Config_ADCAn_ScanGroupx_GetResult ( uint16_t * const buffer );
```

**Remark**  
\( n \) is the unit number, \( x \) is the scan group number.

### Argument(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint16_t * const buffer;</td>
<td>Pointer to area in which to store the results of A/D conversion</td>
</tr>
</tbody>
</table>

### Return value

None.
R_Config_ADCA\textsubscript{n} TH_Group\textsubscript{k} Start

Starts T\&H group hold.

[Syntax]

\begin{verbatim}
void R_Config_ADCA\textsubscript{n} TH_Group\textsubscript{k} Start ( void );
\end{verbatim}

Remark \( n \) is the unit number, \( k \) is the T\&H group number.

[Argument(s)]

None.

[Return value]

None.
### R_Config_ADCAn_TH_Sampling_Start

Starts T&H sampling.

**[Syntax]**

```c
void R_Config_ADCAn_TH_Sampling_Start ( void );
```

**Remark**

$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
### R_Config_ADCan_Create_UserInit

**Performs user-defined initialization relating to the group gcan mode ADCA functions.**

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

**[Syntax]**

```c
void R_Config_ADCan_Create_UserInit ( void );
```

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

**[Argument(s)]**

- None.

**[Return value]**

- None.
**r_Config_ADCAn_error_interrupt**

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

**[Syntax]**

```c
void r_Config_ADCAn_error_interrupt ( void );
```

Remark $n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_Config_ADCAn_scan_groupx_end_interrupt**

Performs processing in response to the scan group end interrupt.

**[Syntax]**

```c
void r_Config_ADCAn_scan_groupx_end_interrupt ( void );
```

**Remark**

$n$ is the unit number, $x$ is the scan group number.

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.3 CSI Slave

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

Table 3.3 API Functions: [CSI Slave]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_Config_CSIGn_Create</td>
<td>Performs initialization necessary to control the CSI slave functions.</td>
</tr>
<tr>
<td>R_Config_CSIGn_Start</td>
<td>Enables CSI.</td>
</tr>
<tr>
<td>R_Config_CSIGn_Stop</td>
<td>Disables CSI.</td>
</tr>
<tr>
<td>R_Config_CSIGn_Send</td>
<td>Start data transmission.</td>
</tr>
<tr>
<td>R_Config_CSIGn_Receive</td>
<td>Start data reception.</td>
</tr>
<tr>
<td>R_Config_CSIGn_Create_UserInit</td>
<td>Performs user-defined initialization relating to the CSI slave functions.</td>
</tr>
<tr>
<td>r_Config_CSIGn_interrupt_send</td>
<td>Processing is performed according to the generation of the send status interrupt.</td>
</tr>
<tr>
<td>r_Config_CSIGn_interrupt_receive</td>
<td>Processing is performed according to the generation of the receive status interrupt.</td>
</tr>
<tr>
<td>r_Config_CSIGn_interrupt_error</td>
<td>Processing is performed according to the occurrence of a communication error interrupt.</td>
</tr>
<tr>
<td>r_Config_CSIGn_callback_sendend</td>
<td>Processing is performed according to the generation of the send status interrupt.</td>
</tr>
<tr>
<td>r_Config_CSIGn_callback_receiveend</td>
<td>Processing is performed according to the generation of the receive status interrupt.</td>
</tr>
<tr>
<td>r_Config_CSIGn_callback_error</td>
<td>Processing is performed according to the occurrence of a communication error interrupt.</td>
</tr>
</tbody>
</table>
R_Config_CSIGN_Create

Performs initialization necessary to control the CSI slave functions.

Remark This API function is called from R_Systeminiit in main () function.

[Syntax]

```c
void R_Config_CSIGN_Create ( void );
```

Remark $n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_Config_CSIGN_Start

Enables CSI.

[Syntax]

```c
void     R_Config_CSIGN_Start ( void );
```

Remark  \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
**R_Config_CSIGN_Stop**

Disables CSI.

**[Syntax]**

```c
void     R_Config_CSIGN_Stop ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_Config_CSIGn_Send**

Start data transmission.

**Remark1.** This API function repeats the 2 byte-level CSIG transmission from the buffer specified in argument `tx_buf` the number of times specified in argument `tx_num`.

**Remark2.** `R_Config_CSIGn_Start` must be called before this API function is called.

**[Syntax]**

```
MD_STATUS    R_Config_CSIGn_Send ( uint16_t* tx_buf, uint16_t tx_num );
```

Remark  

$n$ is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint16_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 specification.</td>
</tr>
</tbody>
</table>
### R_Config_CSIGn_Receive

Start data reception.

Remark 1. This API function performs 2 byte-level CSIG reception the number of times specified by the argument `rx_num`, and stores the data in the buffer specified by the argument `rx_buf`.

Remark 2. Starts after this API function is called, and `R_Config_CSIGn_Start` is then called.

#### [Syntax]

```c
MD_STATUS R_Config_CSIGn_Receive ( uint16_t* rx_buf, uint16_t rx_num);
```

Remark  $n$ is the channel number.

#### [Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint16_t* rx_buf;</td>
<td>Pointer to a buffer to store the received 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 specification</td>
</tr>
</tbody>
</table>
**R_Config_CSIGN_Create_UserInit**

Performs user-defined initialization relating to the CSI slave functions.

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

**[Syntax]**

```c
void R_Config_CSIGN_Create_UserInit ( void );
```

Remark \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
Processing is performed according to the generation of the send status interrupt.

**[Syntax]**

```c
void r_Config_CSIGN_interrupt_send ( void );
```

**Remark**

$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_Config_CSIGN_interrupt_receive**

Processing is performed according to the generation of the receive status interrupt.

**[Syntax]**

```c
void r_Config_CSIGN_interrupt_receive ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
Processing is performed according to the occurrence of a communication error interrupt.

**[Syntax]**
```c
void    r_Config_CSIGN_interrupt_error ( void );
```

**Remark**

\( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_Config_CSIGN_callback_sendend**

Processing is performed according to the generation of the send status interrupt.

**Remark**  This API function is called as the callback routine of interrupt process `r_Config_CSIGN_interrupt_send` corresponding to the CSIG communication interrupt.

**[Syntax]**

```c
void r_Config_CSIGN_callback_sendend ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
Processing is performed according to the generation of the receive status interrupt.

Remark This API function is called as the callback routine of interrupt process 
\texttt{r\_Config\_CSIGN\_interrupt\_receive} corresponding to the CSIG reception interrupt.

**[Syntax]**

```c
void r_Config_CSIGN_callback_receiveend ( void );
```

Remark \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_Config_CSIGN_callback_error**

Processing is performed according to the occurrence of a communication error interrupt.

**Remark**  This API function is called as the callback routine of interrupt process *r_Config_CSIGN_interrupt_error* corresponding to the CSIG error interrupt.

**[Syntax]**

```c
void r_Config_CSIGN_callback_error ( uint32_t err_type );
```

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

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint32_t <em>err_type</em>;</td>
<td>Trigger for CSIG error interrupt.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0000x0x1B : Overrun error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0000x01xB : Parity error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>000010xB : Data consistency check error</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
3.2.4 CSI Master

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

Table 3.4 API Functions: [CSI Master]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_Config_CSI_Gn_Create</td>
<td>Performs initialization necessary to control the CSI master functions.</td>
</tr>
<tr>
<td>R_Config_CSI_Gn_Start</td>
<td>Enables CSI.</td>
</tr>
<tr>
<td>R_Config_CSI_Gn_Stop</td>
<td>Prohibit CSI.</td>
</tr>
<tr>
<td>R_Config_CSI_Gn_Send</td>
<td>Start data transmission.</td>
</tr>
<tr>
<td>R_Config_CSI_Gn_Receive</td>
<td>Start data reception.</td>
</tr>
<tr>
<td>R_Config_CSI_Gn_Create_UserInit</td>
<td>Performs user-defined initialization relating to the CSI master functions.</td>
</tr>
<tr>
<td>r_Config_CSI_Gn_interrupt_send</td>
<td>Processing is performed according to the generation of the send status interrupt.</td>
</tr>
<tr>
<td>r_Config_CSI_Gn_interrupt_receive</td>
<td>Processing is performed according to the generation of the receive status interrupt.</td>
</tr>
<tr>
<td>r_Config_CSI_Gn_interrupt_error</td>
<td>Processing is performed according to the occurrence of a communication error interrupt.</td>
</tr>
<tr>
<td>r_Config_CSI_Gn_callback_sendend</td>
<td>Processing is performed according to the generation of the send status interrupt.</td>
</tr>
<tr>
<td>r_Config_CSI_Gn_callback_receiveend</td>
<td>Processing is performed according to the generation of the receive status interrupt.</td>
</tr>
<tr>
<td>r_Config_CSI_Gn_callback_error</td>
<td>Processing is performed according to the occurrence of a communication error interrupt.</td>
</tr>
</tbody>
</table>
R_Config_CSIGN_Create

Performs initialization necessary to control the CSI master functions.

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

[Syntax]

```c
void R_Config_CSIGN_Create ( void );
```

Remark \( n \) is the channel number.

[Argument(s)]
None.

[Return value] None.
**R_Config_CSIGn_Start**

Enables CSI.

**[Syntax]**

```
void R_Config_CSIGn_Start ( void );
```

**Remark**

$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_Config_CSIGn_Stop**

Disables CSI.

**[Syntax]**

```c
void R_Config_CSIGn_Stop ( void );
```

**Remark**

$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_Config_CSIGN_Send**

Start data transmission.

Remark1. This API function repeats the 2 byte-level CSIG transmission from the buffer specified in argument `tx_buf` the number of times specified in argument `tx_num`.

Remark2. `R_Config_CSIGN_Start` must be called before this API function is called.

Remark3. Depending on the resource used, the arguments are different.

**[Syntax]**

**CSIG case**

```c
MD_STATUS R_Config_CSIGN_Send ( uint16_t* tx_buf, uint16_t tx_num );
```

**CSIH case**

```c
MD_STATUS R_Config_CSIGN_Send ( uint16_t* tx_buf, uint16_t tx_num, uint32_t chip_id );
```

Remark  `n` is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint16_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>
<tr>
<td>I</td>
<td>uint32_t <code>chip_id</code></td>
<td>Slave channel ID (Multiple choices are possible by using &quot;I&quot; operator)</td>
</tr>
<tr>
<td></td>
<td>_CSIH_SELECT_CHIP_n ;</td>
<td>: Channel <code>n</code></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 specification of <code>tx_num</code> or <code>chip_id</code>.</td>
</tr>
</tbody>
</table>
R_Config_CSI Gn_Receive

Start data reception.

Remark1. This API function performs 2 byte-level CSIG reception the number of times specified by the argument \(rx\_num\), and stores the data in the buffer specified by the argument \(rx\_buf\).

Remark2. Starts after this API function is called, and R_Config_CSI Gn_Start is then called.

Remark3. Depending on the resource used, the arguments are different.

[Syntax]

CSIG case

\[
\text{MD\_STATUS} \quad \text{R\_Config\_CSI Gn\_Receive} \ (\text{uint16\_t} \times rx\_buf, \text{uint16\_t} \times rx\_num);
\]

CSIH case

\[
\text{MD\_STATUS} \quad \text{R\_Config\_CSI Gn\_Receive} \ (\text{uint16\_t} \times rx\_buf, \text{uint16\_t} \times rx\_num, \text{uint32\_t} \times chip\_id);
\]

Remark \(n\) is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint16_t \times rx_buf,</td>
<td>Pointer to a buffer to store the received data.</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t \times rx_num;</td>
<td>Total amount of data to receive.</td>
</tr>
<tr>
<td>I</td>
<td>uint32_t \times chip_id,</td>
<td>Slave channel ID (Multiple choices are possible by using &quot;I&quot; operator)</td>
</tr>
<tr>
<td></td>
<td>_CSIH_SELECT_CHIP_n</td>
<td>: Channel n</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 specification of (tx_num) or (chip_id).</td>
</tr>
</tbody>
</table>
R_Config_CSIGn_Create_UserInit

Performs user-defined initialization relating to the CSI master functions.

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

**[Syntax]**
```
void R_Config_CSIGn_Create_UserInit ( void );
```

Remark     `n` is the channel number.

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

**[Return value]**
None.
r_Config_CSIGN_interrupt_send

Processing is performed according to the generation of the send status interrupt.

[Syntax]

```c
void r_Config_CSIGN_interrupt_send ( void );
```

Remark $n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
r_Config_CSIGN_interrupt_receive

Processing is performed according to the generation of the receive status interrupt.

[Syntax]

```c
void r_Config_CSIGN_interrupt_receive ( void );
```

Remark  
$n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
Processing is performed according to the occurrence of a communication error interrupt.

**[Syntax]**

```c
void r_Config_CSIGN_interrupt_error ( void );
```

**Remark**

\( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_Config_CSIGN_callback_sendend**

Processing is performed according to the generation of the send status interrupt.

**Remark** This API function is called as the callback routine of interrupt process `r_Config_CSIGN_interrupt_send` corresponding to the CSIG communication interrupt.

**[Syntax]**

```c
void r_Config_CSIGN_callback_sendend ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
Processing is performed according to the generation of the receive status interrupt.

**Remark** This API function is called as the callback routine of interrupt process

\[r\_Config\_CSIGN\_interrupt\_receive\] corresponding to the CSIG reception interrupt.

**[Syntax]**

```c
void r_Config_CSIGN_callback_receiveend ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
### r_Config_CSIGN_callback_error

Processing is performed according to the occurrence of a communication error interrupt.

**Remark** This API function is called as the callback routine of interrupt process `r_Config_CSIGN_interrupt_error` corresponding to the CSIG error interrupt.

**[Syntax]**

```c
void r_Config_CSIGN_callback_error( uint32_t err_type );
```

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

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint32_t <code>err_type</code>;</td>
<td>Trigger for CSIG error interrupt.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0000x0x1B : Overrun error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0000x01xB : Parity error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>000010xxB : Data consistency check error</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
3.2.5 Interrupt

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

Table 3.5 API Functions: [Interrupt]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_Config_ICU_Create</td>
<td>Performs initialization necessary to control the interrupt functions.</td>
</tr>
<tr>
<td>R_Config_ICU_INTPn_Start</td>
<td>Enables the INTPn interrupts.</td>
</tr>
<tr>
<td>R_Config_ICU_INTPn_Stop</td>
<td>Disables the INTPn interrupts.</td>
</tr>
<tr>
<td>R_Config_ICU_Create_UserInit</td>
<td>Performs user-defined initialization relating to the interrupt functions.</td>
</tr>
<tr>
<td>r_Config_ICU_intpn_interrupt</td>
<td>Performs processing in response to the INTPn interrupt.</td>
</tr>
<tr>
<td>r_Config_ICU_nmi_interrupt</td>
<td>Performs processing in response to the NMI interrupt.</td>
</tr>
</tbody>
</table>
R_Config_ICU_Create

Performs initialization necessary to control the interrupt functions.

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

[Syntax]

```c
void R_Config_ICU_Create ( void );
```

[Argument(s)]
None.

[Return value]
None.
R_Config_ICU_INTPn_Start

Enables the INTPn interrupts.

[Syntax]

```c
void R_Config_ICU_INTPn_Start ( void );
```

Remark  

\[ n \] is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.
R_Config_ICU_INTPn_Stop

Disables the INTPn interrupts.

[Syntax]
void R_Config_ICU_INTPn_Stop ( void );

Remark $n$ is the interrupt factor number.

[Argument(s)]
None.

[Return value]
None.
R_Config_ICU_Create_UserInit

Performs user-defined initialization relating to the interrupt functions.

Remark   This API functions is called as the R_Config_ICU_Create callback routine.

[Syntax]
void     R_Config_ICU_Create_UserInit ( void );

[Argument(s)]
None.

[Return value]
None.
### r_Config_ICU_intp_n_interrupt

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

**[Syntax]**

```c
void r_Config_ICU_intp_n_interrupt ( void );
```

** Remark **

\(n\) is the interrupt factor number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_Config_ICU_nmi_interrupt

Performs processing in response to the NMI interrupt.

[Syntax]
void r_Config_ICU_nmi_interrupt ( void );

[Argument(s)]
None.

[Return value]
None.
3.2.6 Input Interval Timer

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

Table 3.6 API Functions: [Input Interval Timer]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_Config_TAUBn_m_Create</td>
<td>Performs initialization necessary to control the input interval timer functions.</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Start</td>
<td>Starts the count for channel ( m ).</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Stop</td>
<td>Ends the count for channel ( m ).</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Create_UserInit</td>
<td>Performs user-defined initialization relating to the input interval timer functions.</td>
</tr>
<tr>
<td>r_Config_TAUBn_m_interrupt</td>
<td>Performs processing in response to the timer interrupt.</td>
</tr>
</tbody>
</table>
**R_Config_TAUBn_m_Create**

Performs initialization necessary to control the input interval timer functions.

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

**[Syntax]**
```c
void R_Config_TAUBn_m_Create ( void );
```

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

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

**[Return value]**
None.
**R_Config_TAUBn_m_Start**

Starts the count for channel \( m \).

**[Syntax]**

```c
void R_Config_TAUBn_m_Start ( void );
```

**Remark**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_TAUBn_m_Stop

Ends the count for channel \( m \).

[Syntax]

```
void R_Config_TAUBn_m_Stop ( void );
```

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

[Argument(s)]

None.

[Return value]

None.
R_Config_TAUBn_m_Create_UserInit

Performs user-defined initialization relating to the Input Interval Timer functions.

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

[Syntax]

```c
void R_Config_TAUBn_m_Create_UserInit ( void );
```

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

[Argument(s)]

None.

[Return value]

None.
**r_Config_TAUBn_m_interrupt**

Performs processing in response to the timer interrupt.

**[Syntax]**

```c
void r_Config_TAUBn_m_interrupt ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.7 Input Pulse Interval Measurement

Below is a list of API functions output by the Code Generator for input pulse interval measurement use.

Table 3.7 API Functions: [Input Pulse Interval Measurement]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_Config_TAUBn_m_Create</td>
<td>Performs initialization necessary to control the input pulse interval measurement functions.</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Start</td>
<td>Starts the count for channel ( m ).</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Stop</td>
<td>Ends the count for channel ( m ).</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Get_PulseWidth</td>
<td>Reads the input pulse width of the timer.</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Create_UserInit</td>
<td>Performs user-defined initialization relating to the input pulse interval measurement functions.</td>
</tr>
<tr>
<td>r_Config_TAUBn_m_interrupt</td>
<td>Performs processing in response to the timer interrupt.</td>
</tr>
</tbody>
</table>
**R_Config_TAUBn_m_Create**

Performs initialization necessary to control the input pulse interval measurement functions.

Remark  This API function is called from `R_Systeminit` in `main()` function.

**[Syntax]**

```c
void R_Config_TAUBn_m_Create ( void );
```

Remark  `n` is the unit number, `m` is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_Config_TAUBn_m_Start**

Starts the count for channel $m$.

**[Syntax]**

```
void R_Config_TAUBn_m_Start ( void );
```

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

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

**[Return value]**
None.
R_Config_TAUBn_m_Stop

Ends the count for channel $m$.

**[Syntax]**

```c
void R_Config_TAUBn_m_Stop ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_TAUBn_m_Get_PulseWidth

Reads the input pulse width of the timer.

**[Syntax]**

```c
void R_Config_TAUBn_m_Get_PulseWidth ( uint32_t * const width );
```

Remark  

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

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint32_t * const width;</td>
<td>Pointer to area in which to store the results of input pulse width</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
R_Config_TAUBn_m_Create_UserInit

Performs user-defined initialization relating to the input pulse interval measurement functions.

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

[Syntax]

```c
void R_Config_TAUBn_m_Create_UserInit ( void );
```

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

[Argument(s)]
None.

[Return value]
None.
**r_Config_TAUBn_m_interrupt**

Performs processing in response to the timer interrupt.

**[Syntax]**

```c
void r_Config_TAUBn_m_interrupt ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.8 Interval Timer

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

Table 3.8 API Functions: [Interval Timer]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>R_Config_TAUBn_m_Create</code></td>
<td>Performs initialization necessary to control the interval timer functions.</td>
</tr>
<tr>
<td><code>R_Config_TAUBn_m_Start</code></td>
<td>Starts the count for channel <code>m</code>.</td>
</tr>
<tr>
<td><code>R_Config_TAUBn_m_Stop</code></td>
<td>Ends the count for channel <code>m</code>.</td>
</tr>
<tr>
<td><code>R_Config_TAUBn_m_Create_UserInit</code></td>
<td>Performs user-defined initialization relating to the interval timer functions.</td>
</tr>
<tr>
<td><code>r_Config_TAUBn_m_interrupt</code></td>
<td>Performs processing in response to the timer interrupt.</td>
</tr>
</tbody>
</table>
**R_Config_TAUBn_m_Create**

Performs initialization necessary to control the interval timer functions.

Remark  This API function is called from `R_Systeminit` in main () function.

**[Syntax]**

```c
void R_Config_TAUBn_m_Create ( void );
```

Remark  `n` is the unit number, `m` is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_TAUBn_m_Start

Starts the count for channel m.

[Syntax]

    void R_Config_TAUBn_m_Start ( void );

[Argument(s)]

    None.

[Return value]

    None.
R_Config_TAUBn_m_Stop

Ends the count for channel $m$.

[Syntax]

```c
void R_Config_TAUBn_m_Stop ( void );
```

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

[Argument(s)]
None.

[Return value]
None.
**R_Config_TAUBn_m_Create_UserInit**

Performs user-defined initialization relating to the interval timer functions.

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

**[Syntax]**

```c
void R_Config_TAUBn_m_Create_UserInit ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
<table>
<thead>
<tr>
<th>r_Config_TAUB_n_m_interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performs processing in response to the timer interrupt.</td>
</tr>
</tbody>
</table>

**[Syntax]**

```c
void r_Config_TAUBn_m_interrupt ( void );
```

**Remark**

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

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.9 Triangle PWM Output

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

Table 3.9 API Functions: [Triangle PWM Output]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>R_Config_TAUBn_Create</code></td>
<td>Performs initialization necessary to control the triangle PWM output functions.</td>
</tr>
<tr>
<td><code>R_Config_TAUBn_Start</code></td>
<td>Start timer counting.</td>
</tr>
<tr>
<td><code>R_Config_TAUBn_Stop</code></td>
<td>Stops timer counting.</td>
</tr>
<tr>
<td><code>R_Config_TAUBn_Create_UserInit</code></td>
<td>Performs user-defined initialization relating to the triangle PWM output functions.</td>
</tr>
<tr>
<td><code>r_Config_TAUBn_channelsm_interrupt</code></td>
<td>Performs processing in response to the timer interrupt.</td>
</tr>
</tbody>
</table>
R_Config_TAUBn_Create

Performs initialization necessary to control the triangle PWM output functions.

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

[Syntax]

```c
void R_Config_TAUBn_Create ( void );
```

Remark $n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_Config_TAUBn_Start

Start timer counting.

[Syntax]

void R_Config_TAUBn_Start ( void );

Remark  \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
**R_Config_TAUBn_Stop**

Stops timer counting.

**[Syntax]**

```c
void R_Config_TAUBn_Stop ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_TAUBn_Create_UserInit

Performs user-defined initialization relating to the triangle PWM output functions.

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

[Syntax]

```c
void R_Config_TAUBn_Create_UserInit ( void );
```

Remark  \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
**r_Config_TAUBn_channelm_interrupt**

Performs processing in response to the timer interrupt.

**[Syntax]**

```c
void r_Config_TAUBn_channelm_interrupt ( void );
```

**Remark**

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

**[Argument(s)]**

None.

**[Return value]**

None.
### 3.2.10 OS Timer

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

Table 3.10 API Functions: [OS Timer]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_Config_OSTMn_Create</td>
<td>Performs initialization necessary to control the OS timer functions.</td>
</tr>
<tr>
<td>R_Config_OSTMn_Start</td>
<td>Start OS timer count.</td>
</tr>
<tr>
<td>R_Config_OSTMn_Stop</td>
<td>Stop OS timer count.</td>
</tr>
<tr>
<td>R_Config_OSTMn_Set_CompareValue</td>
<td>In interval timer mode, set start value of the down-counter.</td>
</tr>
<tr>
<td></td>
<td>In free-running comparison mode, set value for comparison.</td>
</tr>
<tr>
<td>R_Config_OSTMn_Create_UserInit</td>
<td>Performs user-defined initialization relating to the OS timer functions.</td>
</tr>
<tr>
<td>r_Config_OSTMn_interrupt</td>
<td>Performs processing in response to the OS timer interrupt.</td>
</tr>
</tbody>
</table>
R_Config_OSTMn_Create

Performs initialization necessary to control the OS timer functions.

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

[Syntax]

```c
void R_Config_OSTMn_Create ( void );
```

Remark $n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_Config_OSTMn_Start

Start OS timer count.

[Syntax]

```
void R_Config_OSTMn_Start ( void );
```

Remark $n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_Config_OSTMn_Stop

Stop OS timer count.

[Syntax]

void R_Config_OSTMn_Stop ( void );

Remark  \( n \) is the channel number.

[Argument(s)]
None.

[Return value]
None.
**R_Config_OSTMn_Set_CompareValue**

In interval timer mode, set start value of the down-counter.
In free-running comparison mode, set value for comparison.

**[Syntax]**

```c
void R_Config_OSTMn_Set_CompareValue ( uint32_t value );
```

Remark  
$n$ is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint32_t value;</td>
<td>Start value of the down-counter or comparison value</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
**R_Config_OSTMn_Create_UserInit**

Performs user-defined initialization relating to the OS timer functions.

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

**[Syntax]**

```c
void R_Config_OSTMn_Create_UserInit ( void );
```

**Remark**

$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_Config_OSTMn_interrupt**

Performs processing in response to the OS timer interrupt.

**[Syntax]**

```c
void r_Config_OSTMn_interrupt ( void );
```

**Remark**

$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.11 Port

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

Table 3.11  API Functions: [Port]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_Config_PORT_Create</td>
<td>Performs initialization necessary to control the I/O port functions.</td>
</tr>
<tr>
<td>R_Config_PORT_Create_UserInit</td>
<td>Performs user-defined initialization relating to I/O port functions.</td>
</tr>
</tbody>
</table>
R_Config_PORT_Create

Performs initialization necessary to control the I/O port.

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

[Syntax]
void R_Config_PORT_Create ( void );

[Argument(s)]
None.

[Return value]
None.
**R_Config_PORT_Create_UserInit**

Performs user-defined initialization relating to I/O port.

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

**[Syntax]**

```
void R_Config_PORT_Create_UserInit ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
### 3.2.12 PWM Output

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>R_Config_TAUBn_Create</code></td>
<td>Performs initialization necessary to control the PWM output functions.</td>
</tr>
<tr>
<td><code>R_Config_TAUBn_Start</code></td>
<td>Start timer counting.</td>
</tr>
<tr>
<td><code>R_Config_TAUBn_Stop</code></td>
<td>Stops timer counting.</td>
</tr>
<tr>
<td><code>R_Config_TAUBn_Create_UserInit</code></td>
<td>Performs user-defined initialization relating to the PWM output functions.</td>
</tr>
<tr>
<td><code>r_Config_TAUBn_channelm_interrupt</code></td>
<td>Performs processing in response to the timer interrupt.</td>
</tr>
</tbody>
</table>
**R_Config_TAUB\(n\)_Create**

Performs initialization necessary to control the PWM output functions.

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

**[Syntax]**

```c
void R_Config_TAUBn_Create ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_TAUBn_Start

Start timer counting.

[Syntax]

```c
void R_Config_TAUBn_Start ( void );
```

Remark  $n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
**R_Config_TAUBn_Stop**

Stops timer counting.

**[Syntax]**

```c
void R_Config_TAUBn_Stop ( void );
```

**Remark**

`n` is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_Config_TAUBn_Create_UserInit**

Perform user-defined initialization relating to the PWM output functions.

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

**[Syntax]**

```c
void R_Config_TAUBn_Create_UserInit ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_Config_TAUBn_channelm_interrupt**

Performs processing in response to the timer interrupt.

**[Syntax]**

```c
void r_Config_TAUBn_channelm_interrupt ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.13 Stand-by Controller

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_Config_STBC_Prepare_Stop_Mode</td>
<td>Performs user-defined processing relating to the preparation to start stand-by (STOP mode).</td>
</tr>
<tr>
<td>R_Config_STBC_Start_Stop_Mode</td>
<td>Start stand-by (STOP mode).</td>
</tr>
<tr>
<td>R_Config_STBC_Prepare_Deep_Stop_Mode</td>
<td>Performs user-defined processing relating to the preparation to start stand-by (DeepSTOP mode).</td>
</tr>
<tr>
<td>R_Config_STBC_Start_Deep_Stop_Mode</td>
<td>Start stand-by (DeepSTOP mode).</td>
</tr>
<tr>
<td>R_Config_STBC_Deep_Stop_Loop</td>
<td>Performs wait processing of stand-by (DeepSTOP mode).</td>
</tr>
<tr>
<td>R_Config_STBC_Prepare_Stop_Mode_Set_Peripheral</td>
<td>Performs user-defined processing relating to the preparation (stop peripheral) to start stand-by (STOP mode).</td>
</tr>
<tr>
<td>R_Config_STBC_Prepare_Stop_Mode_Set_Interrupt</td>
<td>Performs user-defined processing relating to the preparation (interrupt control register) to start stand-by (STOP mode).</td>
</tr>
<tr>
<td>R_Config_STBC_Prepare_Stop_Mode_Set_Clock_Mask</td>
<td>Performs user-defined processing relating to the preparation (set the clock stop mask register) to start stand-by (STOP mode).</td>
</tr>
<tr>
<td>R_Config_STBC_Prepare_Stop_Mode_Set_Clock_Source</td>
<td>Performs user-defined processing relating to the preparation (oscillate or stop each clock source) to start stand-by (STOP mode).</td>
</tr>
<tr>
<td>R_Config_STBC_Prepare_Deep_Stop_Mode_Set_Peripheral</td>
<td>Performs user-defined processing relating to the preparation (stop peripheral) to start stand-by (DeepSTOP mode).</td>
</tr>
<tr>
<td>R_Config_STBC_Prepare_Deep_Stop_Mode_Set_Interrupt</td>
<td>Performs user-defined processing relating to the preparation (interrupt control register) to start stand-by (DeepSTOP mode).</td>
</tr>
</tbody>
</table>
R_Config_STBC_Prepare_Stop_Mode

Performs user-defined processing relating to the preparation to start stand-by (STOP mode).

[Syntax]

```c
void R_Config_STBC_Prepare_Stop_Mode ( void );
```

[Argument(s)]

None.

[Return value]

None.
R_Config_STBC_Start_Stop_Mode

Start stand-by (STOP mode).

Remark R_Config_STBC_Prepare_Stop_Mode must be called before this API function is called.

[Syntax]

```c
void R_Config_STBC_Start_Stop_Mode ( void );
```

[Argument(s)]

None.

[Return value]

None.
**R_Config_STBC_Prepare_Deep_Stop_Mode**

Performs user-defined processing relating to the preparation to start stand-by (DeepSTOP mode).

**[Syntax]**

```c
void R_Config_STBC_Prepare_Deep_Stop_Mode ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
**R_Config_STBC_Start_Deep_Stop_Mode**

Start stand-by (DeepSTOP mode).

**Remark**  
*R_Config_STBC_Prepare_Deep_Stop_Mode* must be called before this API function is called.

**[Syntax]**

```c
void R_Config_STBC_Start_Deep_Stop_Mode ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
**R_Config_STBC_Deep_Stop_Loop**

Performs wait processing of stand-by (DeepSTOP mode).

**[Syntax]**

```c
void R_Config_STBC_Deep_Stop_Loop ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
**R_Config_STBC_Prepare_Stop_Mode_Set_Peripheral**

Performs user-defined processing relating to the preparation (stop peripheral) to start stand-by (STOP mode).

**[Syntax]**

```c
void R_Config_STBC_Prepare_Stop_Mode_Set_Peripheral ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_STBC_Prepare_Stop_Mode_Set_Interrupt

Performs user-defined processing relating to the preparation (interrupt control register) to start stand-by (STOP mode).

[Syntax]

```c
void R_Config_STBC_Prepare_Stop_Mode_Set_Interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.
R_Config_STBC_Prepare_Stop_Mode_Set_Clock_Mask

Performs user-defined processing relating to the preparation (set the clock stop mask register) to start stand-by (STOP mode).

[Syntax]

```c
void R_Config_STBC_Prepare_Stop_Mode_Set_Clock_Mask ( void );
```

[Argument(s)]

None.

[Return value]

None.
### R_Config_STBC_Prepare_Stop_Mode_Set_Clock_Source

Performs user-defined processing relating to the preparation (oscillate or stop each clock source) to start stand-by (STOP mode).

**[Syntax]**

```c
void R_Config_STBC_Prepare_Stop_Mode_Set_Clock_Source ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_STBC_Prepae_Dean_Stop_Mode_Set_Peripheral

Performs user-defined processing relating to the preparation (stop peripheral) to start stand-by (DeepSTOP mode).

[Syntax]

```c
void R_Config_STBC_Prepare_Deep_Stop_Mode_Set_Peripheral ( void );
```

[Argument(s)]

None.

[Return value]

None.
R_Config_STBC_Prepare_Deep_Stop_Mode_Set_Interrupt

Performs user-defined processing relating to the preparation (interrupt control register) to start stand-by (DeepSTOP mode).

[Syntax]

```c
void R_Config_STBC_Prepare_Deep_Stop_Mode_Set_Interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.
### UART Interface

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>R_Config_UARTn_Create</code></td>
<td>Performs initialization necessary to control the UART interface functions.</td>
</tr>
<tr>
<td><code>R_Config_UARTn_Start</code></td>
<td>Sets UART communication to standby mode.</td>
</tr>
<tr>
<td><code>R_Config_UARTn_Stop</code></td>
<td>Ends UART communication.</td>
</tr>
<tr>
<td><code>R_Config_UARTn_Send</code></td>
<td>Start UART data transmission.</td>
</tr>
<tr>
<td><code>R_Config_UARTn_Receive</code></td>
<td>Start UART data reception.</td>
</tr>
<tr>
<td><code>R_Config_UARTn_Create_UserInit</code></td>
<td>Performs user-defined initialization relating to the UART interface functions.</td>
</tr>
<tr>
<td><code>r_Config_UARTn_interrupt_send</code></td>
<td>Performs processing in response to the UART communication interrupt.</td>
</tr>
<tr>
<td><code>r_Config_UARTn_interrupt_receive</code></td>
<td>Performs processing in response to the UART reception interrupt.</td>
</tr>
<tr>
<td><code>r_Config_UARTn_interrupt_error</code></td>
<td>Performs processing in response to the UART error interrupt.</td>
</tr>
<tr>
<td><code>r_Config_UARTn_callback_sendend</code></td>
<td>Performs processing in response to the UART communication interrupt.</td>
</tr>
<tr>
<td><code>r_Config_UARTn_callback_receiveend</code></td>
<td>Performs processing in response to the UART reception interrupt.</td>
</tr>
<tr>
<td><code>r_Config_UARTn_callback_error</code></td>
<td>Performs processing in response to the UART error interrupt.</td>
</tr>
</tbody>
</table>
**R_Config_UARTn_Create**

Performs initialization necessary to control the UART interface functions.

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

**[Syntax]**

```c
void R_Config_UARTn_Create ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_UARTn_Start

Sets UART communication to standby mode.

[Syntax]

```c
void R_Config_UARTn_Start ( void );
```

Remark  

$n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
**R_Config_UARTn_Stop**

Ends UART communication.

**[Syntax]**

```c
void R_Config_UARTn_Stop ( void );
```

**Remark**

$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_Config_UARTn_Send**

Start UART data transmission.

Remark1. This API function repeats the 1 byte-level UART transmission from the buffer specified in argument tx_buf the number of times specified in argument tx_num.

Remark2. *R_Config_UARTn_Start* must be called before this API function is called.

**[Syntax]**

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

Remark  

*n* is the channel number.

**[Argument(s)]**

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

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion.</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
<tr>
<td>MD_ERROR</td>
<td>Sending in progress</td>
</tr>
</tbody>
</table>
# R_Config_UARTn_Receive

Start UART data reception.

Remark1. This API function performs 1 byte-level UART reception the number of times specified by the argument `rx_num`, and stores the data in the buffer specified by the argument `rx_buf`.

Remark2. Starts after this API function is called, and `R_Config_UARTn_Start` is then called.

**[Syntax]**

```c
MD_STATUS R_Config_UARTn_Receive ( uint8_t * const rx_buf, uint16_t rx_num );
```

Remark $n$ is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint8_t * const rx_buf;</td>
<td>Pointer to a buffer to store the received 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 specification</td>
</tr>
</tbody>
</table>
**R_Config_UARTn_Create_UserInit**

Performs user-defined initialization relating to the UART interface functions.

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

**[Syntax]**

```c
void R_Config_UARTn_Create_UserInit ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_Config_UARTn_interrupt_send**

Performs processing in response to the UART communication interrupt.

**[Syntax]**

```c
void r_Config_UARTn_interrupt_send ( void );
```

**Remark**

$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_Config_UARTn_interrupt_receive**

Performs processing in response to the UART reception interrupt.

**[Syntax]**

```c
void r_Config_UARTn_interrupt_receive ( void );
```

Remark  
$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_Config_UARTn_interrupt_error**

Performs processing in response to the UART error interrupt.

**[Syntax]**

```c
void r_Config_UARTn_interrupt_error ( void );
```

**Remark**

$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
### r_Config_UARTn_callback_sendend

Performs processing in response to the UART communication interrupt.

**Remark** This API function is called as the callback routine of interrupt process.

**Syntax**

```c
void r_Config_UARTn_callback_sendend ( void );
```

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

**Argument(s)**

None.

**Return value**

None.
**r_Config_UARTn_callback_receiveend**

Performs processing in response to the UART reception interrupt.

**Remark**  This API function is called as the callback routine of interrupt process `r_Config_UARTn_interrupt_receive` corresponding to the UART error interrupt.

**[Syntax]**

```c
void r_Config_UARTn_callback_receiveend ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_Config_UARTn_callback_error**

Performs processing in response to the UART error interrupt.

**Remark**  This API function is called as the callback routine of interrupt process `r_Config_UARTn_interrupt_error` corresponding to the UART error interrupt.

**[Syntax]**

```c
void r_Config_UARTn_callback_error ( uint32_t err_type );
```

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

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
</table>
| I   | uint32_t `err_type`; | Trigger for UART error interrupt  
  0x00xx01B : Bit error  
  0x00x10xB : Overrun error  
  0x001x0xB : Framing error  
  0100xx0xB : Parity error |

**[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 3.15 API Functions: [Watchdog Timer]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_Config_WDTn_Create</td>
<td>Performs initialization necessary to control the watchdog timer functions.</td>
</tr>
<tr>
<td>R_Config_WDTn_Restart</td>
<td>Clears the watchdog timer counter and resumes counting.</td>
</tr>
<tr>
<td>R_Config_WDTn_Create_UserInit</td>
<td>Performs user-defined initialization relating to the watchdog timer functions.</td>
</tr>
<tr>
<td>r_Config_WDTn_interrupt</td>
<td>Performs processing in response to the interval interrupt.</td>
</tr>
</tbody>
</table>
R_Config_WDTn_Create

Performs initialization necessary to control the watchdog timer functions.

Remark  This API function is called from R_SystemInit in main () function.

[Syntax]

```c
void R_Config_WDTn_Create ( void );
```

Remark  \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
**R_Config_WDTn_Restart**

Clears the watchdog timer counter and resumes counting.

**[Syntax]**

```c
void R_Config_WDTn_Restart ( void );
```

**Remark**

$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_Config_WDTn_Create_UserInit**

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

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

**[Syntax]**

```c
void R_Config_WDTn_Create_UserInit ( void );
```

Remark  

$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_Config_WDTn_interrupt**

Performs processing in response to the interval interrupt.

**[Syntax]**

```c
void r_Config_WDTn_interrupt ( void );
```

Remark  
$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.16 Clock Divide

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

Table 3.16 API Functions: [Clock Divide]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_Config_TAUBn_Create</td>
<td>Performs initialization necessary to control the clock divide functions.</td>
</tr>
<tr>
<td>R_Config_TAUBn_Channelm_Start</td>
<td>Starts the count for channel n.</td>
</tr>
<tr>
<td>R_Config_TAUBn_Channelm_Stop</td>
<td>Ends the count for channel n.</td>
</tr>
<tr>
<td>R_Config_TAUBn_Channelm_Get_PulseWidth</td>
<td>Reads the input pulse width of the timer.</td>
</tr>
<tr>
<td>R_Config_TAUBn_Create_UserInit</td>
<td>Performs user-defined initialization relating to the clock divide functions.</td>
</tr>
<tr>
<td>r_Config_TAUBn_channelm_interrupt</td>
<td>Performs processing in response to the timer interrupt.</td>
</tr>
</tbody>
</table>
R_Config_TAUBn_Create

Performs initialization necessary to control the clock divide functions.

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

[Syntax]

```c
void R_Config_TAUBn_Create ( void );
```

Remark \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_Config_TAUBn_Channelm_Start

Starts the count for channel $n$.

**[Syntax]**

```c
void R_Config_TAUBn_Channelm_Start ( void );
```

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

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

**[Return value]**
None.
R_Config_TAUBn_Channelm_Stop

Ends the count for channel $n$.

[Syntax]

```c
void R_Config_TAUBn_Channelm_Stop ( void );
```

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

[Argument(s)]

None.

[Return value]

None.
**R_Config_TAUBn_Channelm_Get_PulseWidth**

Reads the input pulse width of the timer.

**[Syntax]**

```c
void R_Config_TAUBn_Channelm_Get_PulseWidth ( uint32_t * const width );
```

Remark  

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

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint32_t * const width</td>
<td>Pointer to area in which to store the results of input pulse width</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
R_Config_TAUBn_Create_UserInit

Performs user-defined initialization relating to the clock divide functions.

Remark  This API functions is called as the R_Config_TAUBn_Create callback routine.

[Syntax]

```c
void R_Config_TAUBn_Create_UserInit ( void );
```

Remark  \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
*r_Config_TAUB\(n\)_channel\(m\)_interrupt*

Performs processing in response to the timer interrupt.

**[Syntax]**

```c
void r_Config_TAUB\(n\)_channel\(m\)_interrupt ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.17 Data CRC

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_Config_DCRAn_Create</td>
<td>Performs initialization necessary to control the data CRC functions.</td>
</tr>
<tr>
<td>R_Config_DCRAn_Input32bitData</td>
<td>Sets the calculation data for 32 bit width.</td>
</tr>
<tr>
<td>R_Config_DCRAn_Input16bitData</td>
<td>Sets the calculation data for 16 bit width.</td>
</tr>
<tr>
<td>R_Config_DCRAn_Input8bitData</td>
<td>Sets the calculation data for 8 bit width.</td>
</tr>
<tr>
<td>R_Config_DCRAn_GetResult_32bitData</td>
<td>Reads the results of CRC calculation for 32 bit width.</td>
</tr>
<tr>
<td>R_Config_DCRAn_GetResult_16bitData</td>
<td>Reads the results of CRC calculation for 16 bit width.</td>
</tr>
<tr>
<td>R_Config_DCRAn_Create_UserInit</td>
<td>Performs user-defined initialization relating to the data CRC functions.</td>
</tr>
</tbody>
</table>
**R_Config_DCRA\textsubscript{n} \_Create**

Performs initialization necessary to control the data CRC functions.

**Remark** This API function is called from \texttt{R\_Systeminit} in main () function.

**[Syntax]**

```c
void R_Config_DCRA\textsubscript{n} \_Create ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
### R_Config_DCRA\(n\)_Input32bitData

Sets the calculation data for 32 bit width.

**[Syntax]**

```c
void R_Config_DCRA\(n\)_Input32bitData ( const uint32_t * data, uint32_t data_num );
```

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

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>const uint32_t * data;</td>
<td>Pointer to a buffer storing the calculation data.</td>
</tr>
<tr>
<td>I</td>
<td>uint32_t data_num;</td>
<td>Total amount of calculation data.</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
**R_Config_DCRAn_Input16bitData**

Sets the calculation data for 16 bit width.

**[Syntax]**

```c
void R_Config_DCRAn_Input16bitData ( const uint16_t * data, uint32_t data_num );
```

Remark  
$n$ is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>const uint16_t * data;</td>
<td>Pointer to a buffer storing the calculation data.</td>
</tr>
<tr>
<td>I</td>
<td>uint32_t data_num;</td>
<td>Total amount of calculation data.</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
R_Config_DCRAn_Input8bitData

Sets the calculation data for 8 bit width.

**[Syntax]**
```c
void R_Config_DCRAn_Input8bitData ( const uint8_t * data, uint32_t data_num );
```

*Remark*  
$n$ is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>const uint8_t * data;</td>
<td>Pointer to a buffer storing the calculation data.</td>
</tr>
<tr>
<td>I</td>
<td>uint32_t data_num;</td>
<td>Total amount of calculation data.</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
R_Config_DCRAn_GetResult_32bitData

Reads the results of CRC calculation for 32 bit width.

[Syntax]

```c
void R_Config_DCRAn_GetResult_32bitData ( uint32_t * data );
```

Remark \( n \) is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint32_t * data;</td>
<td>Pointer to area in which to store the results of calculation data.</td>
</tr>
</tbody>
</table>

[Return value]

None.
**R_Config_DCRAn_GetResult_16bitData**

Reads the results of CRC calculation for 16 bit width.

**[Syntax]**
```c
void R_Config_DCRAn_GetResult_16bitData ( uint16_t * data );
```

Remark  $n$ is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint16_t * data;</td>
<td>Pointer to area in which to store the results of calculation data.</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
**R_Config_DCRAn_Create_UserInit**

Performs user-defined initialization relating to the data CRC functions.

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

**[Syntax]**

```c
void R_Config_DCRAn_Create_UserInit ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.18 Delay Count

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

Table 3.18 API Functions: [Delay Count]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{R_Config_TAUDn_m_Create}</td>
<td>Performs initialization necessary to control the delay count functions.</td>
</tr>
<tr>
<td>\texttt{R_Config_TAUDn_m_Start}</td>
<td>Starts the count for channel $m$.</td>
</tr>
<tr>
<td>\texttt{R_Config_TAUDn_m_Stop}</td>
<td>Ends the count for channel $m$.</td>
</tr>
<tr>
<td>\texttt{R_Config_TAUDn_m_Create_UserInit}</td>
<td>Performs user-defined initialization relating to the delay count functions.</td>
</tr>
<tr>
<td>\texttt{r_Config_TAUDn_m_interrupt}</td>
<td>Performs processing in response to the timer interrupt.</td>
</tr>
</tbody>
</table>
**R_Config_TAUDn\_m\_Create**

Performs initialization necessary to control the delay count functions.

**Remark** This API function is called from **R\_Systeminit** in main () function.

**[Syntax]**

```c
void R_Config_TAUDn\_m\_Create ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_TAUDn_m_Start

Starts the count for channel \( m \).

**[Syntax]**

```c
void R_Config_TAUDn_m_Start ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_TAUDn_m_Stop

Ends the count for channel $m$.

[Syntax]

```c
void R_Config_TAUDn_m_Stop ( void );
```

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

[Argument(s)]
None.

[Return value]
None.
**R_Config_TAUDn_m_Create_UserInit**

Performs user-defined initialization relating to the delay count functions.

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

**[Syntax]**

```c
void R_Config_TAUDn_m_Create_UserInit ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_Config_TAUDn_m_interrupt

Performs processing in response to the timer interrupt.

**[Syntax]**

```c
void r_Config_TAUDn_m_interrupt ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
### 3.2.19 DMA Controller

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_Config_DMACreate</td>
<td>Performs initialization necessary to control the DMAC functions.</td>
</tr>
<tr>
<td>R_DMACreate</td>
<td>Performs initialization necessary to control the DMAC channel functions.</td>
</tr>
<tr>
<td>R_Config_DMAStart</td>
<td>Enables the DMAC channel transfer.</td>
</tr>
<tr>
<td>R_Config_DMACStop</td>
<td>Disables the DMAC channel transfer.</td>
</tr>
<tr>
<td>R_Config_DMAStartm_Set_SoftwareTrigger</td>
<td>Generates the CMAC channel transfer request.</td>
</tr>
<tr>
<td>R_Config_DMAStartm_Suspend</td>
<td>Suspend DMAC channel transfer.</td>
</tr>
<tr>
<td>R_Config_DMAStartm_Resume</td>
<td>Resume DMAC channel transfer.</td>
</tr>
<tr>
<td>R_Config_DMACreate_UserInit</td>
<td>Performs user-defined initialization relating to the DMAC functions.</td>
</tr>
<tr>
<td>r_Config_DMAinterrupt</td>
<td>Performs processing in response to the DMAC channel interrupt.</td>
</tr>
<tr>
<td>r_Config_DMAcallback_transfer_completion</td>
<td>Performs processing in response to the DMAC transfer end interrupt.</td>
</tr>
<tr>
<td>r_Config_DMAcallback_transfer_count_match</td>
<td>Performs processing in response to the DMAC transfer count match interrupt.</td>
</tr>
</tbody>
</table>
R_Config_DMACn_Create

Performs initialization necessary to control the DMACn functions

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

[Syntax]

```c
void R_Config_DMACy_Create ( void );
```

Remark $n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
**R_DMACnm_Create**

Performs initialization necessary to control the DMAC\(n\) channel\(m\) functions.

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

**[Syntax]**

```c
void R_DMACnm_Create ( void )
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_DMACnm_Start

Enables the DMACn channelm transfer.

[Syntax]

```c
void R_Config_DMACnm_Start ( void );
```

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

[Argument(s)]

None.

[Return value]

None.
R_Config_DMACnm_Stop

Disables the DMACn channelm transfer.

[Syntax]

```c
void R_Config_DMACnm_Stop ( void );
```

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

[Argument(s)]

None.

[Return value]

None.
R_Config_DMACnm_Set_SoftwareTrigger

Generates the CMACn channelm transfer request.

**Syntax**

```c
void R_Config_DMACnm_Set_SoftwareTrigger ( void );
```

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

**Argument(s)**

None.

**Return value**

None.
R_Config_DMACnm_Suspend

Suspend DMAC\textit{n} channel\textit{m} transfer.

**[Syntax]**


def R_Config_DMACnm_Suspend():

**Remark**

\textit{n} is the unit number, \textit{m} is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_Config_DMACnm.Resume**

Resume DMAC\(n\) channel\(m\) transfer.

**[Syntax]**

```c
void R_Config_DMACnmResume ( void );
```

**Remark**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_Config_DMACn_Create_UserInit**

Performs user-defined initialization relating to the DMACn functions.

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

**[Syntax]**

```c
void R_Config_DMACn_Create_UserInit ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_Config_DMACnm_interrupt

Performs processing in response to the DMACn channelm interrupt.

**[Syntax]**

```c
void r_Config_DMACnm_interrupt ( void );
```

Remark  

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_Config_DMACnm_callback_transfer_completion**

Performs processing in response to the DMAC transfer end interrupt.

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

**[Syntax]**

```c
void r_Config_DMACnm_callback_transfer_completion ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_Config_DMACnm_callback_transfer_count_match**

Performs processing in response to the DMAC transfer count match interrupt.

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

**[Syntax]**

```c
void r_Config_DMACnm_callback_transfer_count_match ( void );
```

Remark  `n` is the unit number, `m` is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.20 External Event Count

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

Table 3.20 API Functions: [External Event Count]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_Config_TAUBn_m_Create</td>
<td>Performs initialization necessary to control the external event count functions.</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Start</td>
<td>Starts the count for channel ( m ).</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Stop</td>
<td>Ends the count for channel ( m ).</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Create_UserInit</td>
<td>Performs user-defined initialization relating to the external event count functions.</td>
</tr>
<tr>
<td>r_Config_TAUBn_m_interrupt</td>
<td>Performs processing in response to the timer interrupt.</td>
</tr>
</tbody>
</table>
**R_Config_TAUBn_m_Create**

Performs initialization necessary to control the external event count functions.

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

**[Syntax]**

```c
void R_Config_TAUBn_m_Create ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_Config_TAUBn_m_Start**

Starts the count for channel \( m \).

**[Syntax]**

```c
void R_Config_TAUBn_m_Start ( void );
```

**Remark**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_TAUBn_m_Stop

Ends the count for channel $m$.

[Syntax]

```c
void R_Config_TAUBn_m_Stop ( void );
```

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

[Argument(s)]
None.

[Return value]
None.
**R_Config_TAUBn_m_Create_UserInit**

Performs user-defined initialization relating to the external event count functions.

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

**[Syntax]**

```c
void R_Config_TAUBn_m_Create_UserInit ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_Config_TAUBn_m_interrupt

Performs processing in response to the timer interrupt.

[Syntax]
void r_Config_TAUBn_m_interrupt ( void );

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

[Argument(s)]
None.

[Return value]
None.
### 3.2.21 Input Period Count Detection

Below is a list of API functions output by the Code Generator for input period count detection use.

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_Config_TAUBn_m_Create</td>
<td>Performs initialization necessary to control the input period count detection functions.</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Start</td>
<td>Starts the count for channel ( m ).</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Stop</td>
<td>Ends the count for channel ( m ).</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Get_PulseWidth</td>
<td>Reads the input pulse width of the timer.</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Create_UserInit</td>
<td>Performs user-defined initialization relating to the input period count detection functions.</td>
</tr>
<tr>
<td>r_Config_TAUBn_m_interrupt</td>
<td>Performs processing in response to the timer interrupt.</td>
</tr>
</tbody>
</table>
**R_Config_TAUBn_m_Create**

Performs initialization necessary to control the input period count detection functions.

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

**[Syntax]**

```c
void     R_Config_TAUBn_m_Create ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
Starts the count for channel $m$.

**Syntax**

```c
void     R_Config_TAUBn_m_Start ( void );
```

**Remark**

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

**Argument(s)**

None.

**Return value**

None.
R_Config_TAUBn_m_Stop

Ends the count for channel $m$.

**[Syntax]**

```c
void R_Config_TAUBn_m_Stop ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_TAUBn_m_Get_PulseWidth

Reads the input pulse width of the timer.

[Syntax]

void R_Config_TAUBn_m_Get_PulseWidth ( uint32_t * const width );

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

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint32_t * const width;</td>
<td>Pointer to area in which to store the results of input pulse width.</td>
</tr>
</tbody>
</table>

[Return value]

None.
**R_Config_TAUBn_m_Create_UserInit**

Performs user-defined initialization relating to the input period count detection functions.

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

**[Syntax]**

```c
void R_Config_TAUBn_m_Create_UserInit ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_Config_TAUBn_m_interrupt**

Performs processing in response to the timer interrupt.

**[Syntax]**

```c
void r_Config_TAUBn_m_interrupt ( void );
```

Remark  

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

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.22 Input Position Detection

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

Table 3.22 API Functions: [Input Position Detection]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_Config_TAUBn_m_Create</td>
<td>Performs initialization necessary to control the input position detection functions.</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Start</td>
<td>Starts the count for channel ( m ).</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Stop</td>
<td>Ends the count for channel ( m ).</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Get_PulseWidth</td>
<td>Reads the input pulse width of the timer.</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Create_UserInit</td>
<td>Performs user-defined initialization relating to the input position detection functions.</td>
</tr>
<tr>
<td>r_Config_TAUBn_m_interrupt</td>
<td>Performs processing in response to the timer interrupt.</td>
</tr>
</tbody>
</table>
**R_Config_TAUBn_m_Create**

Performs initialization necessary to control the input position detection functions.

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

**[Syntax]**

```c
void R_Config_TAUBn_m_Create ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_TAUBn_m_Start

Starts the count for channel \( m \).

**[Syntax]**

```c
void R_Config_TAUBn_m_Start ( void );
```

**Remark**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_TAUBn_m_Stop

Ends the count for channel m.

[Syntax]

```c
void R_Config_TAUBn_m_Stop ( void );
```

Remark  

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

[Argument(s)]

None.

[Return value]

None.
R_Config_TAUBn_m_Get_PulseWidth

Reads the input pulse width of the timer.

[Syntax]

```c
void R_Config_TAUBn_m_Get_PulseWidth ( uint32_t * const width );
```

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

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint32_t * const width;</td>
<td>Pointer to area in which to store the results of input pulse width.</td>
</tr>
</tbody>
</table>

[Return value]

None.
**R_Config_TAUBn_m_Create_UserInit**

Performs user-defined initialization relating to the input position detection functions.

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

**[Syntax]**

```c
void R_Config_TAUBn_m_Create_UserInit ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_Config_TAUBn_m_interrupt

Performs processing in response to the timer interrupt.

[Syntax]

```c
void     r_Config_TAUBn_m_interrupt ( void );
```

Remark  

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

[Argument(s)]

None.

[Return value]

None.
3.2.23 Input Pulse Interval Judgment

Below is a list of API functions output by the Code Generator for input pulse interval judgment use.

Table 3.23 API Functions: [Input Pulse Interval Judgment]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_Config_TAUBn_m_Create</td>
<td>Performs initialization necessary to control the input pulse interval judgment functions.</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Start</td>
<td>Starts the count for channel $m$.</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Stop</td>
<td>Ends the count for channel $m$.</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Create_UserInit</td>
<td>Performs user-defined initialization relating to the input pulse interval judgment functions.</td>
</tr>
<tr>
<td>r_Config_TAUBn_m_interrupt</td>
<td>Performs processing in response to the timer interrupt.</td>
</tr>
</tbody>
</table>
**R_Config_TAUBn_m_Create**

Performs initialization necessary to control the input pulse interval judgment functions.

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

**[Syntax]**

```c
void R_Config_TAUBn_m_Create ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_TAUBn_m_Start

Starts the count for channel $m$.

[Syntax]

```c
void R_Config_TAUBn_m_Start ( void );
```

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

[Argument(s)]

None.

[Return value]

None.
**R_Config_TAUBn_m_Stop**

Ends the count for channel \( m \).

**[Syntax]**

```c
void R_Config_TAUBn_m_Stop ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_Config_TAUBn_m_Create_UserInit**

Performs user-defined initialization relating to the input pulse interval judgment functions.

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

**[Syntax]**

```c
void     R_Config_TAUBn_m_Create_UserInit ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_Config_TAUBn_m_interrupt**

Performs processing in response to the timer interrupt.

**[Syntax]**

```c
void r_Config_TAUBn_m_interrupt ( void );
```

Remark  

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

**[Argument(s)]**

None.

**[Return value]**

None.
### 3.2.24 Input Signal Width Judgment

Below is a list of API functions output by the Code Generator for input signal width judgment use.

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_Config_TAUBn_m_Create</td>
<td>Performs initialization necessary to control the input signal width judgment functions.</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Start</td>
<td>Starts the count for channel ( m ).</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Stop</td>
<td>Ends the count for channel ( m ).</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Create_UserInit</td>
<td>Performs user-defined initialization relating to the input signal width judgment functions.</td>
</tr>
<tr>
<td>r_Config_TAUBn_m_interrupt</td>
<td>Performs processing in response to the timer interrupt.</td>
</tr>
</tbody>
</table>
**R_Config_TAUBn_m_Create**

Performs initialization necessary to control the input signal width judgment functions.

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

**[Syntax]**

```c
void R_Config_TAUBn_m_Create ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_Config_TAUBn_m_Start**

Starts the count for channel $m$.

**[Syntax]**

```c
void R_Config_TAUBn_m_Start ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_TAUBn_m_Stop

Ends the count for channel $m$.

**[Syntax]**

```c
void R_Config_TAUBn_m_Stop ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_TAUBn_m_Create_UserInit

Performs user-defined initialization relating to the input signal width judgment functions.

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

[Syntax]

```c
void R_Config_TAUBn_m_Create_UserInit ( void );
```

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

[Argument(s)]
None.

[Return value]
None.
r_Config_TAUBn_m_interrupt

Performs processing in response to the timer interrupt.

[Syntax]

```c
void r_Config_TAUBn_m_interrupt ( void );
```

[Remark] $n$ is the unit number, $m$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
### 3.2.25 Input Signal Width Measurement

Below is a list of API functions output by the Code Generator for input signal width measurement use.

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>R_Config_TAUBn_m_Create</code></td>
<td>Performs initialization necessary to control the input signal width measurement functions.</td>
</tr>
<tr>
<td><code>R_Config_TAUBn_m_Start</code></td>
<td>Starts the count for channel $m$.</td>
</tr>
<tr>
<td><code>R_Config_TAUBn_m_Stop</code></td>
<td>Ends the count for channel $m$.</td>
</tr>
<tr>
<td><code>R_Config_TAUBn_m_Get_PulseWidth</code></td>
<td>Reads the input pulse width of the timer.</td>
</tr>
<tr>
<td><code>R_Config_TAUBn_m_Create_UserInit</code></td>
<td>Performs user-defined initialization relating to the input signal width measurement functions.</td>
</tr>
<tr>
<td><code>r_Config_TAUBn_m_interrupt</code></td>
<td>Performs processing in response to the timer interrupt.</td>
</tr>
</tbody>
</table>
**R_Config_TAUBn_m_Create**

Performs initialization necessary to control the input signal width measurement functions.

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

**[Syntax]**

```c
void R_Config_TAUBn_m_Create ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_TAUBn_m_Start

Starts the count for channel $m$.

[Syntax]

```c
void R_Config_TAUBn_m_Start ( void );
```

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

[Argument(s)]

None.

[Return value]

None.
R_Config_TAUBn_m_Stop

Ends the count for channel \( m \).

[Syntax]

```c
void R_Config_TAUBn_m_Stop ( void );
```

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

[Argument(s)]

None.

[Return value]

None.
**R_Config_TAUBn_m_Get_PulseWidth**

Reads the input pulse width of the timer.

**[Syntax]**

```c
void R_Config_TAUBn_m_Get_PulseWidth ( uint32_t * const width);
```

**Remark**  

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

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint32_t * const width;</td>
<td>Pointer to area in which to store the results of input pulse width.</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
**R_Config_TAUBn_m_Create_UserInit**

Performs user-defined initialization relating to the input signal width measurement functions.

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

**[Syntax]**

```c
void R_Config_TAUBn_m_Create_UserInit ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_Config_TAUBn_m_interrupt
Performs processing in response to the timer interrupt.

[Syntax]

```c
void r_Config_TAUBn_m_interrupt ( void );
```

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

[Argument(s)]

None.

[Return value]

None.
3.2.26 Key Interrupt Function

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

Table 3.26 API Functions: [Key Interrupt Function]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_Config_KEY_Create</td>
<td>Performs initialization necessary to control the key interrupt</td>
</tr>
<tr>
<td></td>
<td>functions.</td>
</tr>
<tr>
<td>R_Config_KEY_Start</td>
<td>Enables the acceptance of the key interrupt.</td>
</tr>
<tr>
<td>R_Config_KEY_Stop</td>
<td>Disables the acceptance of the key interrupt.</td>
</tr>
<tr>
<td>R_Config_KEY_Create_UserInit</td>
<td>Performs user-defined initialization relating to the key interrupt</td>
</tr>
<tr>
<td></td>
<td>functions.</td>
</tr>
<tr>
<td>r_Config_KEY_interrupt</td>
<td>Performs processing in response to the key interrupt.</td>
</tr>
</tbody>
</table>
**R_Config_KEY_Create**

Performs initialization necessary to control the key interrupt functions.

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

**[Syntax]**

```c
void R_Config_KEY_Create ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
**R_Config_KEY_Start**

Enables the acceptance of the key interrupt.

**[Syntax]**

```c
void R_Config_KEY_Start ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_KEY_Stop

Disables the acceptance of the key interrupt.

[Syntax]
void     R_Config_KEY_Stop ( void );

[Argument(s)]
None.

[Return value]
None.
R_Config_KEY_Create_UserInit

Performs user-defined initialization relating to the key interrupt functions.

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

**[Syntax]**

```c
void R_Config_KEY_Create_UserInit ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
r_Config_KEY_interrupt

Performs processing in response to the key interrupt.

[Syntax]

```c
void r_Config_KEY_interrupt ( void );
```

[Argument(s)]

None.

[Return value]

None.
3.2.27 One Pulse Output

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

Table 3.27 API Functions: [One Pulse Output]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_Config_TAUBn_m_Create</td>
<td>Performs initialization necessary to control the one pulse output functions.</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Start</td>
<td>Starts the count for channel $m$.</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Stop</td>
<td>Ends the count for channel $m$.</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Create_UserInit</td>
<td>Performs user-defined initialization relating to the one pulse output functions.</td>
</tr>
<tr>
<td>r_Config_TAUBn_m_interrupt</td>
<td>Performs processing in response to the timer interrupt.</td>
</tr>
</tbody>
</table>
**R_Config_TAUBn_m_Create**

Performs initialization necessary to control the one pulse output functions.

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

**[Syntax]**

```c
void R_Config_TAUBn_m_Create ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_Config_TAUBn_m_Start**

Starts the count for channel $m$.

**[Syntax]**

```c
void R_Config_TAUBn_m_Start ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_TAUBn_m_Stop

Ends the count for channel \( m \).

[Syntax]

void R_Config_TAUBn_m_Stop ( void );

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

[Argument(s)]
None.

[Return value]
None.
**R_Config_TAUBn_m_Create_UserInit**

Performs user-defined initialization relating to the one pulse output functions.

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

**[Syntax]**

```c
void R_Config_TAUBn_m_Create_UserInit ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_Config_TAUBn_m_interrupt**

Performs processing in response to the timer interrupt.

**[Syntax]**

```
void   r_Config_TAUBn_m_interrupt ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.28 One-shot Pulse Output

Below is a list of API functions output by the Code Generator for one-shot pulse output use.

Table 3.28 API Functions: [One-shot Pulse Output]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_Config_TAUBn_m_Create</td>
<td>Performs initialization necessary to control the one-shot pulse output functions.</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Start</td>
<td>Starts the count for channel m.</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Stop</td>
<td>Ends the count for channel m.</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Create_UserInit</td>
<td>Performs user-defined initialization relating to the one-shot pulse output functions.</td>
</tr>
<tr>
<td>r_Config_TAUBn_m_interrupt</td>
<td>Performs processing in response to the timer interrupt.</td>
</tr>
</tbody>
</table>
**R_Config_TAUBn_m_Create**

Performs initialization necessary to control the one-shot pulse output functions.

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

**[Syntax]**

```c
void R_Config_TAUBn_m_Create ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_Config_TAUBn_m_Start**

Starts the count for channel $m$.

**[Syntax]**

```c
void R_Config_TAUBn_m_Start ( void );
```

**Remark**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_Config_TAUBn_m_Stop**

Ends the count for channel \( m \).

**[Syntax]**

```c
void R_Config_TAUBn_m_Stop ( void );
```

**Remark**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_Config_TAUBn_m_Create_UserInit**

Performs user-defined initialization relating to the one-shot pulse output functions.

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

**[Syntax]**

```c
void R_Config_TAUBn_m_Create_UserInit ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_Config_TAUBn_m_interrupt

Performs processing in response to the timer interrupt.

[Syntax]

```c
void r_Config_TAUBn_m_interrupt ( void );
```

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

[Argument(s)]

None.

[Return value]

None.
3.2.29 Overflow Interrupt Output (Input period count detecting)

Below is a list of API functions output by the Code Generator for overflow interrupt output (input period count detecting) use.

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_Config_TAUBn_m_Create</td>
<td>Performs initialization necessary to control the overflow interrupt output (input period count detecting) functions.</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Start</td>
<td>Starts the count for channel $m$.</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Stop</td>
<td>Ends the count for channel $m$.</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Get_PulseWidth</td>
<td>Reads the input pulse width of the timer.</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Create_UserInit</td>
<td>Performs user-defined initialization relating to the overflow interrupt output (input period count detecting) functions.</td>
</tr>
<tr>
<td>r_Config_TAUBn_m_interrupt</td>
<td>Performs processing in response to the timer interrupt.</td>
</tr>
</tbody>
</table>
**R_Config_TAUBn_m_Create**

Performs initialization necessary to control the overflow interrupt output (input period count detecting) functions.

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

**[Syntax]**

```c
void R_Config_TAUBn_m_Create ( void );
```

Remark  

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_TAUBn_m_Start

Starts the count for channel \( m \).

[Syntax]

```c
void R_Config_TAUBn_m_Start ( void );
```

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

[Argument(s)]

None.

[Return value]

None.
R_Config_TAUBn_m_Stop

Ends the count for channel \( m \).

**[Syntax]**

```c
void R_Config_TAUBn_m_Stop ( void );
```

**[Remark]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_TAUB\textsubscript{n\_m} \textunderscore Get\_PulseWidth

Reads the input pulse width of the timer.

[Syntax]

\texttt{void \ R\_Config\_TAUBn\_m\_Get\_PulseWidth ( uint32\_t * const width );}

Remark \hspace{1em} \( n \) is the unit number, \( m \) is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint32_t * const width;</td>
<td>Pointer to area in which to store the results of input pulse width.</td>
</tr>
</tbody>
</table>

[Return value]

None.
Performs user-defined initialization relating to the overflow interrupt output (input period count detecting) functions.

Remark This API functions is called as the R_Config_TAUBn_m_Create callback routine.

**[Syntax]**

```c
void R_Config_TAUBn_m_Create_UserInit ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_Config_TAUBn_m_interrupt**

Performs processing in response to the timer interrupt.

**[Syntax]**

```c
void r_Config_TAUBn_m_interrupt ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.30 Overflow Interrupt Output (Width measurement)

Below is a list of API functions output by the Code Generator for overflow interrupt output (width measurement) use.

Table 3.30 API Functions: [Overflow Interrupt Output (Width measurement)]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_Config_TAUBn_m_Create</td>
<td>Performs initialization necessary to control the overflow interrupt output (width measurement) functions.</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Start</td>
<td>Starts the count for channel $m$.</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Stop</td>
<td>Ends the count for channel $m$.</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Get_PulseWidth</td>
<td>Reads the input pulse width of the timer.</td>
</tr>
<tr>
<td>R_Config_TAUBn_m_Create_UserInit</td>
<td>Performs user-defined initialization relating to the overflow interrupt output (width measurement) functions.</td>
</tr>
<tr>
<td>r_Config_TAUBn_m_interrupt</td>
<td>Performs processing in response to the timer interrupt.</td>
</tr>
</tbody>
</table>
**R_Config_TAUB\_n\_m\_Create**

Performs initialization necessary to control the overflow interrupt output (width measurement) functions.

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

**[Syntax]**

```c
void R_Config_TAUBn_m_Create ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_TAUBn_m_Start

Starts the count for channel $m$.

**[Syntax]**

```c
void R_Config_TAUBn_m_Start ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_Config_TAUB\(n\)\(_{m}\)\_Stop**

Ends the count for channel \(m\).

**[Syntax]**

```c
void R_Config_TAUB\(n\)\(_{m}\)\_Stop ( void );
```

Remark

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_Config_TAUBn_m_Get_PulseWidth**

Reads the input pulse width of the timer.

**Syntax**

```c
void R_Config_TAUBn_m_Get_PulseWidth ( uint32_t * const width );
```

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

**Argument(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint32_t * const width;</td>
<td>Pointer to area in which to store the results of input pulse width.</td>
</tr>
</tbody>
</table>

**Return value**

None.
R_Config_TAUBn_m_Create_UserInit

Performs user-defined initialization relating to the overflow interrupt output (width measurement) functions.

Remark This API functions is called as the R_Config_TAUBn_m_Create callback routine.

[Syntax]

void R_Config_TAUBn_m_Create_UserInit ( void );

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

[Argument(s)]

None.

[Return value]

None.
**r_Config_TAUBn_m_interrupt**

Performs processing in response to the timer interrupt.

**[Syntax]**

```c
void r_Config_TAUBn_m_interrupt ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
### 3.2.31 IIC Master Mode

Below is a list of API functions output by the Code Generator for IIC master mode use.

**Table 3.31 API Functions: [IIC Master Mode]**

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_Config_RIICn_Create</td>
<td>Performs initialization necessary to control the IIC master mode functions.</td>
</tr>
<tr>
<td>R_Config_RIICn_Start</td>
<td>Enables IIC.</td>
</tr>
<tr>
<td>R_Config_RIICn_Stop</td>
<td>Disables IIC.</td>
</tr>
<tr>
<td>R_Config_RIICn_Master_Send</td>
<td>Starts master transmission.</td>
</tr>
<tr>
<td>R_Config_RIICn_Master_Send_Without_Stop</td>
<td>Starts master transmission (with no stop condition being issued at the end of transmission).</td>
</tr>
<tr>
<td>R_Config_RIICn_Master_Receive</td>
<td>Starts master reception.</td>
</tr>
<tr>
<td>R_Config_RIICn_StartCondition</td>
<td>Issues a start condition.</td>
</tr>
<tr>
<td>R_Config_RIICn_StopCondition</td>
<td>Issues a stop condition.</td>
</tr>
<tr>
<td>R_Config_RIICn_Create_UserInit</td>
<td>Performs user-defined initialization relating to the IIC master mode functions.</td>
</tr>
<tr>
<td>r_Config_RIICn_error_interrupt</td>
<td>Executes processing in response to communication error interrupts or communication event generation interrupts.</td>
</tr>
<tr>
<td>r_Config_RIICn_receive_interrupt</td>
<td>Executes processing in response to receive data full interrupts.</td>
</tr>
<tr>
<td>r_Config_RIICn_transmit_interrupt</td>
<td>Executes processing in response to transmit data empty interrupts.</td>
</tr>
<tr>
<td>r_Config_RIICn_transmitend_interrupt</td>
<td>Executes processing in response to transmit end interrupts.</td>
</tr>
<tr>
<td>r_Config_RIICn_callback_transmitend</td>
<td>Executes processing specific to the detection of a stop condition in master transmission among the sources of communication error interrupts or communication event generation interrupts. In master transmission without issuing a stop condition, this API function executes processing in response to transmit end interrupts.</td>
</tr>
<tr>
<td>r_Config_RIICn_callback_receiveend</td>
<td>Executes processing specific to the detection of a stop condition in master reception among the sources of communication error interrupts or communication event generation interrupts.</td>
</tr>
<tr>
<td>r_Config_RIICn_callback_receiveerror</td>
<td>Executes processing specific to the detection of a loss in arbitration, NACK, or timeout among the sources of communication error interrupts or communication event generation interrupts.</td>
</tr>
</tbody>
</table>
**R_Config_RIICn_Create**

Performs initialization necessary to control the IIC master mode functions.

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

**[Syntax]**

```c
void R_Config_RIICn_Create ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_Config_RIIcn_Start**

Enables IIC.

**[Syntax]**

```c
void R_Config_RIIcn_Start ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
Disables IIC.

[Syntax]

```c
void     R_Config_RIICn_Stop ( void );
```

Remark  
\( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
**R_Config_RIIcn_Master_Send**

Starts master transmission.

**Remark1.** This API function executes the master transmission of the slave address specified by the argument `adr` and the R/W# bit to slave devices, and then repeats the master transmission of single bytes of data from the buffer specified by the argument `tx_buf` the number of times specified by the argument `tx_num`.

**Remark2.** This API function internally calls `R_Config_RIIcn_StartCondition` to start master transmission.

**Remark3.** A stop condition is issued in `r_Config_RIIcn_transmitend_interrupt` to stop master transmission.

**Remark4.** Calling `R_Config_RIIcn_Start` is required before this API function is called to execute master transmission.

**[Syntax]**

```c
MD_STATUS    R_Config_RIIcn_Master_Send ( uint16_t adr, uint8_t * const tx_buf, uint16_t tx_num );
```

Remark n is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint16_t adr;</td>
<td>Slave address.</td>
</tr>
<tr>
<td>I</td>
<td>uint8_t * const tx_buf;</td>
<td>Pointer to the buffer where the data to be transmitted are stored.</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t tx_num;</td>
<td>Number of bytes to be transmitted.</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>The bus is busy.</td>
</tr>
<tr>
<td>MD_ERROR2</td>
<td>The specification of argument <code>adr</code> is invalid.</td>
</tr>
</tbody>
</table>
**R_Config_RIICn_Master_Send_Without_Stop**

Starts master transmission (with no stop condition being issued at the end of transmission).

**Remark1.** This API function executes the master transmission of the slave address specified by the argument *adr* and the R/W# bit to slave devices, and then repeats the master transmission of single bytes of data from the buffer specified by the argument *tx_buf* the number of times specified by the argument *tx_num*.

**Remark2.** This API function internally calls `R_Config_RIICn_StartCondition` to start master transmission.

**Remark3.** `r_Config_RIICn_transmitend_interrupt` does not issue a stop condition to stop master transmission.

**Remark4.** Calling `R_Config_RIICn_Start` is required before this API function is called to execute master transmission.

**[Syntax]**

```c
MD_STATUS R_Config_RIICn_Master_Send_Without_Stop ( uint16_t *adr, uint8_t * const *tx_buf, uint16_t *tx_num );
```

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</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>uint8_t * const *tx_buf;</code></td>
<td>Pointer to the buffer where the data to be transmitted are stored</td>
</tr>
<tr>
<td>I</td>
<td><code>uint16_t *tx_num;</code></td>
<td>Number of bytes to be transmitted</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>The bus is busy.</td>
</tr>
<tr>
<td>MD_ERROR2</td>
<td>The specification of argument <em>adr</em> is invalid.</td>
</tr>
</tbody>
</table>
R_Config_RIICn_Master_Receive

Starts master reception.

Remark1. This API function executes the master transmission of the slave address specified by the argument \textit{adr} to slave devices, and then repeats the master reception of single bytes of data the number of times specified by the argument \textit{rx\_num}, storing the data in the buffer specified by the argument \textit{rx\_buf}.

Remark2. This API function internally calls \texttt{R_Config_RIICn\_StartCondition} to start master reception.

Remark3. A stop condition is issued in \texttt{r\_Config\_RIICn\_receive\_interrupt} to stop master reception.

Remark4. Calling \texttt{R_Config_RIICn\_Start} is required before this API function is called to execute master reception.

[Syntax]

\begin{verbatim}
MD\_STATUS R_Config_RIICn\_Master\_Receive ( uint16\_t \textit{adr}, uint8\_t * const \textit{rx\_buf}, uint16\_t \textit{rx\_num} );
\end{verbatim}

Remark \(n\) is the channel number.

[Argument(s)]

\begin{center}
\begin{tabular}{|c|c|l|}
\hline
I/O & Argument(s) & Description \\
\hline
I & \textbf{uint16\_t \textit{adr};} & Slave address \\
O & \textbf{uint8\_t * const \textit{rx\_buf};} & Pointer to the buffer where the received data are to be stored \\
I & \textbf{uint16\_t \textit{rx\_num};} & Number of bytes to be received \\
\hline
\end{tabular}
\end{center}

[Return value]

\begin{center}
\begin{tabular}{|c|l|}
\hline
Macro & Description \\
\hline
MD\_OK & Normal completion. \\
MD\_ERROR1 & The bus is busy. \\
MD\_ERROR2 & The specification of argument \textit{adr} is invalid. \\
\hline
\end{tabular}
\end{center}
**R_Config_RIIICn_StartCondition**

Issues a start condition.

Remarks

In use with the RIIC module, a call of this API function generates a communication error/communication event generation interrupt, after which `r_Config_RIIICn_error_interrupt` is called.

**[Syntax]**

```c
void R_Config_RIIICn_StartCondition ( void );
```

Remarks

`n` is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_RIIcn_StopCondition

Issues a stop condition.

Remark In use with the RIIC module, a call of this API function generates a communication
error/communication event generation interrupt, after which
r_Config_RIIcn_error_interrupt is called.

[Syntax]

```c
void R_Config_RIIcn_StopCondition ( void );
```

Remark $n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_Config_RIICn_Create_UserInit

Performs user-defined initialization relating to the IIC master mode functions.

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

[Syntax]

```c
void R_Config_RIICn_Create_UserInit ( void );
```

Remark  \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
**r_Config_RIICn_error_interrupt**

Executes processing in response to communication error interrupts or communication event generation interrupts.

**[Syntax]**

```c
void r_Config_RIICn_error_interrupt ( void );
```

Remark  

\( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_Config_RIICh_receive_interrupt**

Executes processing in response to receive data full interrupts.

**[Syntax]**

```c
void r_Config_RIICh_receive_interrupt ( void );
```

**Remark**

*n* is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_Config_RIICh Transmit Interrupt**

Executes processing in response to transmit data empty interrupts.

**[Syntax]**

```c
void r_Config_RIICh_transmit_interrupt ( void );
```

**Remark**

$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_Config_RIIcn_transmitend_interrupt**

Executes processing in response to transmit end interrupts.

**[Syntax]**

```c
void r_Config_RIIcn_transmitend_interrupt ( void );
```

**Remark**

$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_Config_RIICn_callback_transmitend**

Executes processing specific to the detection of a stop condition in master transmission among the sources of communication error interrupts or communication event generation interrupts.

**Remark1.** This API function is called from `r_Config_RIICn_error_interrupt` as a callback routine.
**Remark2.** To execute master transmission, call `R_Config_RIICn_Master_Send`.

In master transmission without issuing a stop condition, this API function executes processing in response to transmit end interrupts

**Remark3.** This API function is called from `r_Config_RIICn_transmitend_interrupt` as a callback routine.
**Remark4.** To execute master transmission, call `R_Config_RIICn_Master_Send_Without_Stop`.

**[Syntax]**

```c
void r_Config_RIICn_callback_transmitend ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_Config_RIIcn_callback_receiveend**

Executes processing specific to the detection of a stop condition in master reception among the sources of communication error interrupts or communication event generation interrupts.

**Remark 1.** This API function is called from `r_Config_RIIcn_error_interrupt` as a callback routine.

**Remark 2.** To execute master reception, call `R_Config_RIIcn_Master_Receive`.

**[Syntax]**

```c
void r_Config_RIIcn_callback_receiveend ( void );
```

**Remark**  

$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
### `r_Config_RIICn_callback_receiveerror`

Executes processing specific to the detection of a loss in arbitration, NACK, or timeout among the sources of communication error interrupts or communication event generation interrupts.

**Remark**  
This API function is called from `r_Config_RIICn_error_interrupt` as a callback routine.

**[Syntax]**

```c
void r_Config_RIICn_callback_receiveerror ( MD_STATUS status );
```

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

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
</table>
| I   | MD_STATUS `status`; | Interrupt sources  
|     |   MD_ERROR1 : Detection of loss in arbitration  
|     |   MD_ERROR2 : Detection of timeout  
|     |   MD_ERROR3 : Detection of NACK |

**[Return value]**

None.
### 3.2.32 IIC Slave Mode

Below is a list of API functions output by the Code Generator for IIC slave mode use.

#### Table 3.32 API Functions: [IIC Slave Mode]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>R_Config_RIICn_Create</code></td>
<td>Performs initialization necessary to control the IIC slave mode functions.</td>
</tr>
<tr>
<td><code>R_Config_RIICn_Start</code></td>
<td>Enables IIC.</td>
</tr>
<tr>
<td><code>R_Config_RIICn_Stop</code></td>
<td>Disables IIC.</td>
</tr>
<tr>
<td><code>R_Config_RIICn_Slave_Send</code></td>
<td>Starts slave transmission.</td>
</tr>
<tr>
<td><code>R_Config_RIICn_Slave_Receive</code></td>
<td>Starts slave reception.</td>
</tr>
<tr>
<td><code>R_Config_RIICn_StartCondition</code></td>
<td>Issues a start condition.</td>
</tr>
<tr>
<td><code>R_Config_RIICn_StopCondition</code></td>
<td>Issues a stop condition.</td>
</tr>
<tr>
<td><code>R_Config_RIICn_Create_UserInit</code></td>
<td>Performs user-defined initialization relating to the IIC slave mode functions.</td>
</tr>
<tr>
<td><code>r_Config_RIICn_error_interrupt</code></td>
<td>Executes processing in response to communication error interrupts.</td>
</tr>
<tr>
<td><code>r_Config_RIICn_receive_interrupt</code></td>
<td>Executes processing in response to receive data full interrupts.</td>
</tr>
<tr>
<td><code>r_Config_RIICn_transmit_interrupt</code></td>
<td>Executes processing in response to transmit data empty interrupts.</td>
</tr>
<tr>
<td><code>r_Config_RIICn_transmitend_interrupt</code></td>
<td>Executes processing in response to transmit end interrupts.</td>
</tr>
<tr>
<td><code>r_Config_RIICn_callback_transmitend</code></td>
<td>Executes processing specific to the detection of a stop condition in slave transmission among the sources of communication error interrupts or communication event generation interrupts.</td>
</tr>
<tr>
<td><code>r_Config_RIICn_callback_receiveend</code></td>
<td>Executes processing specific to the detection of a stop condition in slave reception among the sources of communication error interrupts or communication event generation interrupts.</td>
</tr>
<tr>
<td><code>r_Config_RIICn_callback_receiveerror</code></td>
<td>Executes processing specific to the detection of a loss in arbitration, NACK, or timeout among the sources of communication error interrupts or communication event generation interrupts.</td>
</tr>
</tbody>
</table>
### R_Config_RIICn_Create

Perform initialization necessary to control the IIC slave mode functions.

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

**[Syntax]**

```c
void R_Config_RIICn_Create ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_RIICn_Start

Enables IIC.

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

Remark $n$ is the channel number.

[Argument(s)]
None.

[Return value]
None.
R_Config_RIIcn_Stop

Disables IIC.

[Syntax]

```c
void R_Config_RIIcn_Stop ( void );
```

Remark  \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
**R_Config_RIICn_Slave_Send**

Starts slave transmission.

Remark1. This API function repeats the slave transmission of single bytes of data from the buffer specified by the argument `tx_buf` the number of times specified by the argument `tx_num`.

Remark2. Calling `R_Config_RIICn_Start` is required before this API function is called to execute transmission.

**[Syntax]**

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

Remark  

$n$ is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint8_t * const tx_buf;</td>
<td>Pointer to the buffer where the data to be transmitted are stored.</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t tx_num;</td>
<td>Number of bytes to be transmitted.</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion.</td>
</tr>
</tbody>
</table>
R_Config_RIICn_Slave_Receive

Starts slave reception.

Remark1. This API function repeats the slave reception of single bytes of data the number of times specified by the argument rx_num, storing the data in the buffer specified by the argument rx_buf.

Remark2. Calling R_Config_RIICn_Start is required before this API function is called to execute slave reception.

[Syntax]

```
MD_STATUS    R_Config_RIICn_Slave_Receive ( uint8_t * const rx_buf, uint16_t rx_num);
```

Remark    $n$ is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint8_t * const rx_buf;</td>
<td>Pointer to the buffer where the received data are to be stored.</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t rx_num;</td>
<td>Number of bytes to be received.</td>
</tr>
</tbody>
</table>

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion.</td>
</tr>
</tbody>
</table>
R_Config_RIICn_StartCondition

Issues a start condition.

Remark In use with the RIIC module, a call of this API function generates a communication error/communication event generation interrupt, after which r_Config_RIICn_error_interrupt is called.

[Syntax]

```c
void R_Config_RIICn_StartCondition ( void );
```

Remark $n$ is the channel number.

[Argument(s)]
None.

[Return value]
None.
R_Config_RIICn_StopCondition

Issues a stop condition.

Remark In use with the RIIC module, a call of this API function generates a communication error/communication event generation interrupt, after which r_Config_RIICn_error_interrupt is called.

[Syntax]

```c
void R_Config_RIICn_StopCondition ( void );
```

Remark \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
**R_Config_RIICn_Create_UserInit**

Performs user-defined initialization relating to the IIC slave mode functions.

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

**[Syntax]**

```c
void R_Config_RIICn_Create_UserInit( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_Config_RIICn_error_interrupt**

Executes processing in response to communication error interrupts or communication event generation interrupts.

**[Syntax]**

```c
void r_Config_RIICn_error_interrupt ( void );
```

Remark  
$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_Config_RIICh.receive_interrupt**

Executes processing in response to receive data full interrupts.

**[Syntax]**

```c
void r_Config_RIICh.receive_interrupt ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
Executes processing in response to transmit data empty interrupts.

**[Syntax]**

```c
void r_Config_RIICn_transmit_interrupt ( void );
```

Remark  
$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_Config_RIICh Transmit End Interrupt

Executes processing in response to transmit end interrupts.

[Syntax]

```c
void r_Config_RIICh_TransmitEnd Interrupt ( void );
```

[Remark] $n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
**r_Config_RIIcn_callback_transmitend**

Executes processing specific to the detection of a stop condition in slave transmission among the sources of communication error interrupts or communication event generation interrupts.

**Remark1.** This API function is called from `r_Config_RIIcn_error_interrupt` as a callback routine.

**Remark2.** To execute slave transmission, call `R_Config_RIIcn_Slave_Send`.

**[Syntax]**

```c
void r_Config_RIIcn_callback_transmitend ( void );
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_Config_RIIcn_callback_receiveend**

Executes processing specific to the detection of a stop condition in slave reception among the sources of communication error interrupts or communication event generation interrupts.

Remark1. This API function is called from `r_Config_RIIcn_error_interrupt` as a callback routine.
Remark2. To execute slave transmission, call `R_Config_RIIcn_Slave_Receive`.

**[Syntax]**

```c
void r_Config_RIIcn_callback_receiveend ( void );
```

Remark  
$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_Config_RIICn_callback_receiveerror**

Executes processing specific to the detection of a loss in arbitration, NACK, or timeout among the sources of communication error interrupts or communication event generation interrupts.

Remark This API function is called from `r_Config_RIICn_error_interrupt` as a callback routine.

**[Syntax]**

```c
void     r_Config_RIICn_callback_receiveerror ( MD_STATUS status );
```

Remark  

\[ n \] is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>MD_STATUS status;</td>
<td>Interrupt sources</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_ERROR1 : Detection of loss in arbitration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_ERROR2 : Detection of timeout</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_ERROR3 : Detection of NACK</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
3.2.33 Real-Time Clock

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

Table 3.33 API Functions: [Real-Time Clock]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_Config_RTCAn_Create</td>
<td>Performs initialization necessary to control the real-time clock.</td>
</tr>
<tr>
<td>R_Config_RTCAn_Start</td>
<td>Starts the count of the real-time clock (year, month, weekday, day, hour, minute, second).</td>
</tr>
<tr>
<td>R_Config_RTCAn_Stop</td>
<td>Ends the count of the real-time clock (year, month, weekday, day, hour, minute, second).</td>
</tr>
<tr>
<td>R_Config_RTCAn_Set_HourSystem</td>
<td>Sets the clock type (12-hour or 24-hour clock) of the real-time clock.</td>
</tr>
<tr>
<td>R_Config_RTCAn_Get_CounterBufferValue</td>
<td>Read counter buffer register.</td>
</tr>
<tr>
<td>R_Config_RTCAn_Get_CounterDirectValue</td>
<td>Read the counter register directly.</td>
</tr>
<tr>
<td>R_Config_RTCAn_Set_CounterValue</td>
<td>Sets the counter value (year, month, weekday, day, hour, minute, second) of the real-time clock.</td>
</tr>
<tr>
<td>R_Config_RTCAn_Get_SubCounterValue</td>
<td>Read the buffer register of the sub-counter.</td>
</tr>
<tr>
<td>R_Config_RTCAn_Set_ErrorCorrectionValue</td>
<td>Set the error correction value.</td>
</tr>
<tr>
<td>R_Config_RTCAn_Set_SubCounterCompareValue</td>
<td>Set the comparison value of the sub-counter.</td>
</tr>
<tr>
<td>R_Config_RTCAn_Get_AlarmValue</td>
<td>Reads the alarm conditions (weekday, hour, minute).</td>
</tr>
<tr>
<td>R_Config_RTCAn_Set_AlarmValue</td>
<td>Sets the alarm conditions (weekday, hour, minute).</td>
</tr>
<tr>
<td>R_Config_RTCAn_Set_AlarmOn</td>
<td>Starts the alarm interrupt function.</td>
</tr>
<tr>
<td>R_Config_RTCAn_Set_AlarmOff</td>
<td>Ends the alarm interrupt function.</td>
</tr>
<tr>
<td>R_Config_RTCAn_Set_ConstPeriodInterruptOn</td>
<td>Sets the cycle of the periodic interrupts, then starts the periodic interrupt function.</td>
</tr>
<tr>
<td>R_Config_RTCAn_Set_ConstPeriodInterruptOff</td>
<td>Ends the periodic interrupt function.</td>
</tr>
<tr>
<td>R_Config_RTCAn_Set_1secondInterruptOn</td>
<td>Starts the 1 second interrupt function.</td>
</tr>
<tr>
<td>R_Config_RTCAn_Set_1secondInterruptOff</td>
<td>Ends the 1 second interrupt function.</td>
</tr>
<tr>
<td>R_Config_RTCAn_Set_RTCA1HZOn</td>
<td>Enables output of the correction clock (1 Hz) to the RTC1HZ pin.</td>
</tr>
<tr>
<td>R_Config_RTCAn_Set_RTCA1HZOff</td>
<td>Disables output of the correction clock (1 Hz) to the RTC1HZ pin.</td>
</tr>
<tr>
<td>R_Config_RTCAn_Create_UserInit</td>
<td>Performs user-defined initialization relating to the real-time clock.</td>
</tr>
<tr>
<td>r_Config_RTCAn_interrupt_alarm</td>
<td>Performs processing in response to the alarm interrupt.</td>
</tr>
<tr>
<td>r_Config_RTCAn_interrupt_periodic</td>
<td>Performs processing in response to the periodic interrupt.</td>
</tr>
<tr>
<td>r_Config_RTCAn_interrupt_1second</td>
<td>Performs processing in response to the 1 second interrupt.</td>
</tr>
</tbody>
</table>
**R_Config_RTCAn_Create**

Performs initialization necessary to control the real-time clock.

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

**[Syntax]**

```c
void R_Config_RTCAn_Create ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_RTCn_Start

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

[Syntax]

```c
void R_Config_RTCn_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.
**R_Config_RTCAn_Stop**

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

**[Syntax]**

```c
void R_Config_RTCAn_Stop ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_RTCan_Set_HourSystem

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

**[Syntax]**

```c
MD_STATUS R_Config_RTCan_Set_HourSystem ( rtc_hour_system_t hour_system );
```

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>rtc_hour_system_t hour_system;</td>
<td>Clock type</td>
</tr>
</tbody>
</table>

- HOUR12 : 12-hour clock
- HOUR24 : 24-hour clock

**[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_BUSY1</td>
<td>Executing count process (before change to setting)</td>
</tr>
<tr>
<td>MD_BUSY2</td>
<td>Stopping count process (after change to setting)</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
R_Config_RTCAn_Get_CounterBufferValue

Read counter buffer register.

[Syntax]

```
MD_STATUS R_Config_RTCAn_Get_CounterBufferValue ( rtc_counter_value_t * const counter_read_val);
```

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>rtc_counter_value_t * const counter_read_val</td>
<td>Pointer to the structure storing the read count value.</td>
</tr>
</tbody>
</table>

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

```c
typedef struct
{
    uint8_t sec; /* second */
    uint8_t min; /* minute */
    uint8_t hour; /* hour */
    uint8_t day; /* day */
    uint8_t week; /* weekday (0: sunday, 6: saturday) */
    uint8_t month; /* month */
    uint8_t year; /* year */
} rtc_counter_value_t;
```

[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_BUSY1</td>
<td>Executing count process (before change to setting).</td>
</tr>
<tr>
<td>MD_BUSY2</td>
<td>Stopping count process (after change to setting).</td>
</tr>
</tbody>
</table>
R_Config_RTCAn_Get.CounterDirectValue

Read the counter register directly.

[Syntax]

```
MD_STATUS R_Config_RTCAn_Get.CounterDirectValue ( rtc_counter_value_t * const counter_read_val);
```

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>rtc_counter_value_t * const</td>
<td>Pointer to the structure storing the read count value</td>
</tr>
</tbody>
</table>

Remarks
For structure rtc_counter_value_t, see R_Config_RTCAn_Get.CounterBufferValue.

[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_ERROR</td>
<td>Mismatch between first and second read values.</td>
</tr>
</tbody>
</table>
**R_Config_RTCAn_Set_CounterValue**

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

**[Syntax]**

```c
MD_STATUS R_Config_RTCAn_Set_CounterValue ( rtc_counter_value_t counter_write_val );
```

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>rtc_counter_value_t counter_write_val</td>
<td>Counter value (year, month, weekday, day, hour, minute, second).</td>
</tr>
</tbody>
</table>

**Remark** For structure rtc_counter_value_t, see R_Config_RTCAn_Get_CounterBufferValue.

**[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_BUSY1</td>
<td>Executing count process (before change to setting).</td>
</tr>
<tr>
<td>MD_BUSY2</td>
<td>Stopping count process (after change to setting).</td>
</tr>
</tbody>
</table>
R_Config_RTCAn_Get_SubCounterValue

Read the buffer register of the sub-counter.

[Syntax]

```
MD_STATUS R_Config_RTCAn_Get_SubCounterValue ( uint32_t * const subcounter_read_val );
```

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>uint32_t * const subcounter_read_val;</td>
<td>Pointer to the structure storing the read sub-count value</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_BUSY</td>
<td>Transfer incomplete.</td>
</tr>
</tbody>
</table>
**R_Config_RTCA\text{\textsubscript{n}}_Set_ErrorCorrectionValue**

Set the error correction value.

**[Syntax]**

```c
MD_STATUS R_Config_RTCA\text{\textsubscript{n}}_Set_ErrorCorrectionValue ( uint8_t * const errorcorrection_write_val );
```

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint8_t * const errorcorrection_write_val;</td>
<td>Pointer to the structure storing the error correction value.</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_BUSY1</td>
<td>Operation stopped status.</td>
</tr>
<tr>
<td>MD_BUSY2</td>
<td>Write incomplete.</td>
</tr>
</tbody>
</table>
**R_Config_RTCAn_Set_SubCounterCompareValue**

Set the comparison value of the sub-counter.

**[Syntax]**

```c
MD_STATUS R_Config_RTCAn_Set_SubCounterCompareValue ( uint32_t * const subcompare_write_val );
```

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint32_t * const</td>
<td>Pointer to the structure storing the comparison value.</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_BUSY1</td>
<td>Operation stopped status.</td>
</tr>
<tr>
<td>MD_BUSY2</td>
<td>Write incomplete.</td>
</tr>
</tbody>
</table>
R_Config_RTCAn_Get_AlarmValue

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

**Syntax**

\[
\text{void R_Config_RTCAn\_Get\_AlarmValue ( rtc\_alarm\_value\_t * const alarm\_val );}
\]

**Argument(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>rtc_alarm_value_t * const alarm_val</td>
<td>Pointer to structure in which to store the conditions being read.</td>
</tr>
</tbody>
</table>

**Remark**

Below is shown the structure rtc\_alarm\_value\_t (alarm conditions).

```c
typedef struct {
    uint8_t alarmwm;  /* minute */
    uint8_t alarmwh;  /* hour */
    uint8_t alarmww;  /* weekday */
} rtc\_alarm\_value\_t;
```

**Return value**

None.
R_Config_RTCAn_Set_AlarmValue

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

[Syntax]
void R_Config_RTCAn_Set_AlarmValue ( rtc_alarm_value_t alarm_val );

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>rtc_alarm_value_t</td>
<td>Alarm conditions (weekday, hour, minute)</td>
</tr>
</tbody>
</table>

Remark For structure rtc_alarm_value_t, see R_Config_RTCAn_Get_AlarmValue.

[Return value]
None.
R_Config_RTCAn_Set_AlarmOn

Starts the alarm interrupt function.

**[Syntax]**

```c
void R_Config_RTCAn_Set_AlarmOn ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_RTCAn_Set_AlarmOff

Ends the alarm interrupt function.

Syntax

```c
void R_Config_RTCAn_Set_AlarmOff ( void );
```

Argument(s)

None.

Return value

None.
### R_Config_RTCa\text{n} Set_ConstPeriodInterruptOn

Sets the cycle of the periodic interrupts, then starts the periodic interrupt function.

**[Syntax]**

```
MD_STATUS R_Config_RTCa\text{n} Set_ConstPeriodInterruptOn ( rtc_int_period_t period );
```

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>rtc_int_period_t period</td>
<td>Interrupt INTRTC cycle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QUARTERSEC : 0.25 seconds</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HALFSEC : 0.5 seconds</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ONESEC : 1 second</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ONEMIN : 1 minute</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ONEHOUR : 1 hour</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ONEDAY : 1 day</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ONEMONTH : 1 month</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 specification.</td>
</tr>
</tbody>
</table>
R_Config_RTCAn_Set_ConstPeriodInterruptOff

Ends the periodic interrupt function.

[Syntax]

void R_Config_RTCAn_Set_ConstPeriodInterruptOff ( void );

[Argument(s)]

None.

[Return value]

None.
**R_Config_RTCAn_Set_1secondInterruptOn**

Starts the 1 second interrupt function.

**[Syntax]**

```c
void R_Config_RTCAn_Set_1secondInterruptOn ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
**R_Config_RTCAn_Set_1secondInterruptOff**

Ends the 1 second interrupt function.

**[Syntax]**

```
void     R_Config_RTCAn_Set_1secondInterruptOff ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_Config_RTCAn_Set_RTCA1HZOn

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

[Syntax]

void R_Config_RTCAn_Set_RTCA1HZOn ( void );

[Argument(s)]

None.

[Return value]

None.
Disables output of the correction clock (1 Hz) to the RTC1HZ pin.

**[Syntax]**

```c
void R_Config_RTCAN_Set_RTCA1HZOff ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
**R_Config_RTCAn_Create_UserInit**

Performs user-defined initialization relating to the real-time clock.

**Remark**  This API functions is called as the **R_Config_RTCAn_Create** callback routine.

**[Syntax]**

```c
void R_Config_RTCAn_Create_UserInit ( void );
```

**[Argument(s)]**

None.

**[Return value]**

None.
r_Config_RTCAn_interrupt_alarm

Performs processing in response to the alarm interrupt.

[Syntax]

void r_Config_RTCAn_interrupt_alarm ( void );

[Argument(s)]

None.

[Return value]

None.
### r_Config_RTCAn_interrupt_periodic

Performs processing in response to the periodic interrupt.

**Syntax**

```c
void     r_Config_RTCAn_interrupt_periodic ( void );
```

**Argument(s)**

None.

**Return value**

None.
r_Config_RTCAn_interrupt_1second

Performs processing in response to the 1 second interrupt.

[Syntax]

```c
void r_Config_RTCAn_interrupt_1second ( void );
```

[Argument(s)]
None.

[Return value]
None.
## Revision Record

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>July 13, 2018</td>
<td>First Edition issued</td>
</tr>
</tbody>
</table>