Smart Configurator
User's Manual: RH850 API Reference

RENESAS MCU
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 be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.

5. 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 any third parties arising from such alteration, modification, copying or reverse engineering.

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

   - “Standard”: Computers, office equipment, communications equipment, test and measurement equipment, audio and visual equipment, home electronic appliances, machine tools, personal electronic equipment, industrial robots, etc.
   - “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

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

7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising from any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS, WILL BE FREE FROM ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION ("Vulnerability Issues"). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.

8. 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.

9. 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 resistant 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.

10. 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 a product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document. 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.

11. 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.

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

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

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Trademarks

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

Contact information

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

www.renesas.com/contact/

© 2023 Renesas Electronics Corporation. All rights reserved.
General Precautions in the Handling of Micro-processing Unit and Microcontroller Unit Products

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

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

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

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

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

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

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

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

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

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

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

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

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 ... XXX
  Hexadecimal ... 0xXXXX

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

Corporate Headquarters ............................................................................................................. 1

Contact information ................................................................................................................ 1

Trademarks ............................................................................................................................... 1

1. GENERAL .......................................................................................................................... 7
   1.1 Overview ....................................................................................................................... 7
   1.2 Features ....................................................................................................................... 7

2. OUTPUT FILES ................................................................................................................. 8
   2.1 Description ................................................................................................................... 8

3. API FUNCTIONS ............................................................................................................... 29
   3.1 Overview ..................................................................................................................... 29
   3.2 Function Reference ..................................................................................................... 30
      3.2.1 General .................................................................................................................. 32
      3.2.2 A/D Converter ..................................................................................................... 86
      3.2.3 CSI Slave ............................................................................................................ 125
      3.2.4 CSI Master ......................................................................................................... 140
      3.2.5 Interrupt .............................................................................................................. 156
      3.2.6 Input Interval Timer ............................................................................................ 173
      3.2.7 Input Pulse Interval Measurement ...................................................................... 180
      3.2.8 Interval Timer ....................................................................................................... 188
      3.2.9 Triangle PWM Output ......................................................................................... 195
      3.2.10 Triangle PWM Output With Dead Time ............................................................. 202
      3.2.11 OS Timer ............................................................................................................ 209
      3.2.12 Port ..................................................................................................................... 216
      3.2.13 PWM Output ..................................................................................................... 219
      3.2.14 Stand-by Controller ........................................................................................... 226
      3.2.15 UART Interface .................................................................................................. 245
      3.2.16 Watchdog Timer ................................................................................................. 260
      3.2.17 Clock Divide ....................................................................................................... 269
      3.2.18 Data CRC ........................................................................................................... 276
      3.2.19 Delay Count ....................................................................................................... 295
      3.2.20 DMA Controller ................................................................................................. 302
      3.2.21 External Event Count ......................................................................................... 324
      3.2.22 Input Period Count Detection ............................................................................ 331
3.2.23  Input Position Detection ............................................................................................................... 339
3.2.24  Input Pulse Interval Judgment .................................................................................................. 347
3.2.25  Input Signal Width Judgment .................................................................................................. 354
3.2.26  Input Signal Width Measurement ............................................................................................... 361
3.2.27  Key Interrupt Function .............................................................................................................. 369
3.2.28  One Pulse Output ...................................................................................................................... 375
3.2.29  One-shot Pulse Output .............................................................................................................. 382
3.2.30  Overflow Interrupt Output (Input period count detecting) ....................................................... 390
3.2.31  Overflow Interrupt Output (Width measurement) ...................................................................... 397
3.2.32  IIC Master Mode ....................................................................................................................... 404
3.2.33  IIC Slave Mode ......................................................................................................................... 422
3.2.34  SCI3 Clock Synchronous Mode ................................................................................................. 439
3.2.35  SCI3 Asynchronous Mode ......................................................................................................... 454
3.2.36  MSPI Master .............................................................................................................................. 471
3.2.37  MSPI Slave ............................................................................................................................... 489
3.2.38  Real-Time Clock ....................................................................................................................... 507
3.2.39  Error Control Module ............................................................................................................... 532
3.2.40  GTM Clock ................................................................................................................................. 549
3.2.41  Time Base Unit ........................................................................................................................ 553
3.2.42  TIM Bit Compression Mode ..................................................................................................... 558
3.2.43  TIM Gated Periodic Sampling Mode ........................................................................................ 574
3.2.44  TIM Input Prescaler Mode ....................................................................................................... 589
3.2.45  TIM Input Event Mode ............................................................................................................. 604
3.2.46  TIM Pulse Integration Mode ................................................................................................... 618
3.2.47  TIM PWM Measurement Mode ............................................................................................... 632
3.2.48  TIM Serial Shift Mode ............................................................................................................. 646
3.2.49  ATOM Signal Output Mode Compare .................................................................................... 665
3.2.50  ATOM Signal Output Mode Immediate .................................................................................. 671
3.2.51  ATOM Signal Output Mode PWM ........................................................................................ 676
3.2.52  ATOM Signal Output Mode Serial ........................................................................................ 682
3.2.53  Dead Time Module .................................................................................................................. 688
3.2.54  DTS Controller ......................................................................................................................... 691
3.2.55  ADC Boundary Flag Generator ............................................................................................... 701

Revision Record ...................................................................................................................................... 707
1. GENERAL
This chapter gives an overview of the driver code generator (hereafter abbreviated as Code Generator) of the Smart Configurator.

1.1 Overview
This 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 for xxxx. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */

“xxxx” is changed for different user code:
- “global” – user can add global variables and functions
- “function” – user can add functions declaration in .h file
- “user init” – user can add initializing code
- Interrupt function name – user can add service routine code
- “adding” – user can add functions in .c file

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 (1/21)

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>File Name</th>
<th>API Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>r_cg_main.c</td>
<td>main R_MAIN_UserInit</td>
</tr>
<tr>
<td></td>
<td>r_cg_systeminit.c</td>
<td>R_Systeminit</td>
</tr>
<tr>
<td></td>
<td>r_cg_cgc.c</td>
<td>R_CGC_Create</td>
</tr>
<tr>
<td></td>
<td>r_cg_cgc_user.c</td>
<td>R_CGC_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td>r_cg_cgc.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_smc_clock_info.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_intvector.c</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_intvector_PEk.c</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_intc_PEk.c</td>
<td>R_Interrupt.Initialize_ForPE</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_cg_ad.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_abfg.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_ad_common.c</td>
<td>R_ADC_SyncStart R_ADC_TimerSyncStart</td>
</tr>
<tr>
<td></td>
<td>r_cg_ad_common.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_ad_air.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_csig.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_csih.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_dcra.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_dma.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>General</td>
<td>r_cg_dma_common.c</td>
<td>R_DMAC_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DMA_Suspend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DMA_Resume</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_PDMA_Suspend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_PDMA_Resume</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DMAC_Start_Error_Interrupt_PEk</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DMAC_Stop_Error_Interrupt_PEk</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DMA_Start_Transfer_Error_Interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_DMA_Stop_Transfer_Error_Interrupt</td>
</tr>
<tr>
<td></td>
<td>r_cg_dma_common_user.c</td>
<td>r_sdmac_address_error_interrupt_pek</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_dtsg_transfer_end_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_dtsg_transfer_count_match_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_dmac_error_interrupt_pek</td>
</tr>
<tr>
<td>r_cg_dma_common.h</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>r_cg_dtm.h</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>r_cg_dts.h</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>r_cg_dts_common.c</td>
<td>R_DTS_Suspend</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R_DTS_Resume</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R_DTSg_Start_Transfer_End_Interrupt</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R_DTSg_Stop_Transfer_End_Interrupt</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R_DTSg_Start_Transfer_Count_Match INTERRUPT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R_DTSg_Stop_Transfer_Count_Match_INTERRUPT</td>
<td></td>
</tr>
<tr>
<td>r_cg_dts_common_user.c</td>
<td>r_dtsg_transfer_end_interrupt</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_dtsg_transfer_count_match_interrupt</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_dts_transfer_error_interrupt_pek</td>
<td></td>
</tr>
<tr>
<td>r_cg_dts_common.h</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>r_cg_ecm.h</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>r_cg_gtm_clock.h</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>r_cg_gtm_common.h</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>r_cg_gtm_common.c</td>
<td>R_GTM_Start_Interrupt_Error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R_GTM_Stop_Interrupt_Error</td>
<td></td>
</tr>
<tr>
<td>r_cg_gtm_common_user.c</td>
<td>r_gtm_error_interrupt</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>General</td>
<td>r_cg_atom.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_atom_common.c</td>
<td>R_ATOMn_Start_Interrupt_IRQk</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ATOMn_Stop_Interrupt_IRQk</td>
</tr>
<tr>
<td></td>
<td>r_cg_atom_common_user.c</td>
<td>r_atomn_irqk_interrupt</td>
</tr>
<tr>
<td></td>
<td>r_cg_intc.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_kcrc.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_key.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_mspi.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_mspi_common.c</td>
<td>R_MSPI_Master_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_MSPI_Start_Interrupt_MSPIinTX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_MSPI_Stop_Interrupt_MSPIinTX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_MSPI_Start_Interrupt_MSPIinRX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_MSPI_Stop_Interrupt_MSPIinRX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_MSPI_Start_Interrupt_MSPIinFE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_MSPI_Stop_Interrupt_MSPIinFE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_MSPI_Start_Interrupt_MSPIinERR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_MSPI_Stop_Interrupt_MSPIinERR</td>
</tr>
<tr>
<td></td>
<td>r_cg_mspi_common_user.c</td>
<td>r_mspin_interrupt_send</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_mspin_interrupt_receive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_mspin_interrupt_frameend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_mspin_interrupt_error</td>
</tr>
<tr>
<td></td>
<td>r_cg_mspi_common.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_ostm.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_port.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_ric.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_rtc.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_sci3.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_stbc.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_taub.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_taud.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_tauj.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_tbu.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_tim.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_uart.h</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_cg_wdt.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>r_smc_intprg.c</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_smc_entry.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 2.4  Output File List (4/21)

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>File Name</th>
<th>API Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>A/D Converter</td>
<td>&lt;Config_ADCXn&gt;.c</td>
<td>R_&lt;Config_ADCXn&gt;<em>Create                R</em>&lt;Config_ADCXn&gt;_Halt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_ADCXn&gt;<em>SelfDiagnostic_VoltageCircuitOn         R</em>&lt;Config_ADCXn&gt;_SelfDiagnostic_VoltageCircuitOff</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_ADCXn&gt;<em>ScanGroupx_Start         R</em>&lt;Config_ADCXn&gt;_ScanGroupx_OperationOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_ADCXn&gt;<em>ScanGroupx_OperationOff          R</em>&lt;Config_ADCXn&gt;_ScanGroupx_GetResult</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_ADCXn&gt;<em>ScanGroupx_GetPWMDiagResult        R</em>&lt;Config_ADCXn&gt;_ScanGroupx_GetFloatingPointResult</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_ADCXn&gt;<em>SetMultiplexerCommand         R</em>&lt;Config_ADCXn&gt;_TH_Groupk_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_ADCXn&gt;<em>TH_Sampling_Start           R</em>&lt;Config_ADCXn&gt;_TH_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_ADCXn&gt;<em>ScanGroupx_TimerStart         R</em>&lt;Config_ADCXn&gt;_ScanGroupx_TimerStop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_ADCXn&gt;<em>VoltageDivider_Start         R</em>&lt;Config_ADCXn&gt;_VoltageDivider_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_ADCXn&gt;<em>StrongPullDown_Start         R</em>&lt;Config_ADCXn&gt;_StrongPullDown_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_ADCXn&gt;<em>SGDiag_Start                R</em>&lt;Config_ADCXn&gt;_SGDiag_OperationOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_ADCXn&gt;<em>SGDiag_OperationOff          R</em>&lt;Config_ADCXn&gt;_SGDiag_GetResult</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_ADCXn&gt;<em>SGDiag_GetSubtractionResult    R</em>&lt;Config_ADCXn&gt;_ASF_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_ADCXn&gt;<em>ASF_Stop                   R</em>&lt;Config_ADCXn&gt;_ASF_GetResult</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_ADCXn&gt;<em>ASF_GetRealTimeResult        R</em>&lt;Config_ADCXn&gt;_user.c</td>
</tr>
<tr>
<td></td>
<td>&lt;Config_ADCXn&gt;_user.c</td>
<td>R_&lt;Config_ADCXn&gt;<em>Create_UserInit          r</em>&lt;Config_ADCXn&gt;_error_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_ADCXn&gt;<em>scan_groupx_end_interrupt    r</em>&lt;Config_ADCXn&gt;_sg_diag_end_interrupt</td>
</tr>
<tr>
<td></td>
<td>&lt;Config_ADCXn&gt;_h</td>
<td>r_&lt;Config_ADCXn&gt;<em>mpx_request_interrupt       r</em>&lt;Config_ADCXn&gt;_asf_channelk_end_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_ADCXn&gt;_monitor_error_interrupt</td>
</tr>
</tbody>
</table>
### Table 2.5 Output File List (5/21)

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>File Name</th>
<th>API Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSI Slave</td>
<td><code>&lt;Config_CSIxn&gt;</code>.c</td>
<td>R_&lt;Config_CSIxn&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_CSIxn&gt;_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_CSIxn&gt;_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_CSIxn&gt;_Send</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_CSIxn&gt;_Receive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_CSIxn&gt;_Send_Receive</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_CSIxn&gt;_user.c</code></td>
<td>R_&lt;Config_CSIxn&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_CSIxn&gt;_interrupt_send</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_CSIxn&gt;_interrupt_receive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_CSIxn&gt;_interrupt_error</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_CSIxn&gt;.h</code></td>
<td>—</td>
</tr>
<tr>
<td>CSI Master</td>
<td><code>&lt;Config_CSIxn&gt;</code>.c</td>
<td>R_&lt;Config_CSIxn&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_CSIxn&gt;_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_CSIxn&gt;_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_CSIxn&gt;_Send</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_CSIxn&gt;_Receive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_CSIxn&gt;_Send_Receive</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_CSIxn&gt;_user.c</code></td>
<td>R_&lt;Config_CSIxn&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_CSIxn&gt;_interrupt_send</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_CSIxn&gt;_interrupt_receive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_CSIxn&gt;_interrupt_error</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_CSIxn&gt;.h</code></td>
<td>—</td>
</tr>
<tr>
<td>Interrupt</td>
<td><code>&lt;Config_INTC&gt;</code>.c</td>
<td>R_&lt;Config_INTC&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_INTC&gt;_INTPn_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_INTC&gt;_INTPn_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_INTC&gt;_IRQn_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_INTC&gt;_IRQn_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_INTC&gt;_NMI_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_INTC&gt;_NMI_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_INTC&gt;_SINTn_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_INTC&gt;_SINTn_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_INTC&gt;_SINTn_TriggerOn</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_INTC&gt;_user.c</code></td>
<td>R_&lt;Config_INTC&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_INTC&gt;_intp_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_INTC&gt;_intp_interrupt_pek</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_INTC&gt;_irqn_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_INTC&gt;_nmi_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_INTC&gt;_sint1_interrupt_pek</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_INTC&gt;.h</code></td>
<td>—</td>
</tr>
</tbody>
</table>
### Table 2.6  Output File List (6/21)

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>File Name</th>
<th>API Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Interval Timer</strong></td>
<td><code>&lt;Config_TAUXn_m&gt;.c</code></td>
<td>R_&lt;Config_TAUXn_m&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TAUXn_m&gt;_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TAUXn_m&gt;_Stop</td>
</tr>
<tr>
<td><code>&lt;Config_TAUXn_m&gt;_user.c</code></td>
<td>R_&lt;Config_TAUXn_m&gt;_Create_UserInit</td>
<td></td>
</tr>
<tr>
<td><code>&lt;Config_TAUXn_m&gt;.h</code></td>
<td>r_&lt;Config_TAUXn_m&gt;_interrupt</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_&lt;Config_TAUXn_m&gt;_interrupt_pek</td>
<td></td>
</tr>
<tr>
<td><strong>Input Pulse Interval</strong></td>
<td><code>&lt;Config_TAUXn_m&gt;.c</code></td>
<td>R_&lt;Config_TAUXn_m&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TAUXn_m&gt;_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TAUXn_m&gt;_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TAUXn_m&gt;_Get_PulseWidth</td>
</tr>
<tr>
<td><code>&lt;Config_TAUXn_m&gt;_user.c</code></td>
<td>R_&lt;Config_TAUXn_m&gt;_Create_UserInit</td>
<td></td>
</tr>
<tr>
<td><code>&lt;Config_TAUXn_m&gt;.h</code></td>
<td>r_&lt;Config_TAUXn_m&gt;_interrupt</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_&lt;Config_TAUXn_m&gt;_interrupt_pek</td>
<td></td>
</tr>
<tr>
<td><strong>Interval Timer</strong></td>
<td><code>&lt;Config_TAUXn_m&gt;.c</code></td>
<td>R_&lt;Config_TAUXn_m&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TAUXn_m&gt;_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TAUXn_m&gt;_Stop</td>
</tr>
<tr>
<td><code>&lt;Config_TAUXn_m&gt;_user.c</code></td>
<td>R_&lt;Config_TAUXn_m&gt;_Create_UserInit</td>
<td></td>
</tr>
<tr>
<td><code>&lt;Config_TAUXn_m&gt;.h</code></td>
<td>r_&lt;Config_TAUXn_m&gt;_interrupt</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r_&lt;Config_TAUXn_m&gt;_interrupt_pek</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>Triangle PWM Output</td>
<td><code>&lt;Config_TAU&lt;Xn&gt;.c</code></td>
<td>R_&lt;Config_TAU&lt;Xn&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TAU&lt;Xn&gt;_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TAU&lt;Xn&gt;_Stop</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAU&lt;Xn&gt;_user.c</code></td>
<td>R_&lt;Config_TAU&lt;Xn&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_TAU&lt;Xn&gt;_channelm_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_TAU&lt;Xn&gt;_channelm_interrupt_pek</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAU&lt;Xn&gt;.h</code></td>
<td>—</td>
</tr>
<tr>
<td>Triangle PWM Output</td>
<td><code>&lt;Config_TAU&lt;Xn&gt;.c</code></td>
<td>R_&lt;Config_TAU&lt;Xn&gt;_Create</td>
</tr>
<tr>
<td>With Deadtime</td>
<td></td>
<td>R_&lt;Config_TAU&lt;Xn&gt;_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TAU&lt;Xn&gt;_Stop</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAU&lt;Xn&gt;_user.c</code></td>
<td>R_&lt;Config_TAU&lt;Xn&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_TAU&lt;Xn&gt;_channelm_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_TAU&lt;Xn&gt;_channelm_interrupt_pek</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAU&lt;Xn&gt;.h</code></td>
<td>—</td>
</tr>
<tr>
<td>OS Timer</td>
<td><code>&lt;Config_OSTM&lt;n&gt;.c</code></td>
<td>R_&lt;Config_OSTM&lt;n&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_OSTM&lt;n&gt;_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_OSTM&lt;n&gt;_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_OSTM&lt;n&gt;_Set_CompareValue</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_OSTM&lt;n&gt;_user.c</code></td>
<td>R_&lt;Config_OSTM&lt;n&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_OSTM&lt;n&gt;_interrupt</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_OSTM&lt;n&gt;.h</code></td>
<td>—</td>
</tr>
<tr>
<td>Port</td>
<td><code>&lt;Config_PORT&gt;.c</code></td>
<td>R_&lt;Config_PORT&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>—</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_PORT&gt;_user.c</code></td>
<td>R_&lt;Config_PORT&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_PORT&gt;.h</code></td>
<td>—</td>
</tr>
<tr>
<td>PWM Output</td>
<td><code>&lt;Config_TAU&lt;Xn&gt;.c</code></td>
<td>R_&lt;Config_TAU&lt;Xn&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TAU&lt;Xn&gt;_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TAU&lt;Xn&gt;_Stop</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAU&lt;Xn&gt;_user.c</code></td>
<td>R_&lt;Config_TAU&lt;Xn&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_TAU&lt;Xn&gt;_channelm_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_TAU&lt;Xn&gt;_channelm_interrupt_pek</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAU&lt;Xn&gt;.h</code></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>Stand-by Controller</td>
<td><code>&lt;Config_STBC&gt;.c</code></td>
<td>R_&lt;Config_STBC&gt;_Prepare_Stop_Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_STBC&gt;_Start_Stop_Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_STBC&gt;_Prepare_Deep_Stop_Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_STBC&gt;_Start_Deep_Stop_Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_STBC&gt;_Deep_Stop_Loop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_STBC&gt;_Set_Module_XXXX_Standby_Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_STBC&gt;_Cancel_Module_XXXX_Standby_Mode</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_STBC&gt;_user.c</code></td>
<td>R_&lt;Config_STBC&gt;_Prepare_Stop_Mode_Set_Peripheral</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_STBC&gt;_Prepare_Stop_Mode_Set_Interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_STBC&gt;_Prepare_Stop_Mode_Set_Clock_Mask</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_STBC&gt;_Prepare_Stop_Mode_Set_CPUCLK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_STBC&gt;_Prepare_Deep_Stop_Mode_Set_Peripheral</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_STBC&gt;_Prepare_Deep_Stop_Mode_Set_Interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_STBC&gt;_Prepare_Deep_Stop_Mode_Set_Clock_Mask</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_STBC&gt;_Prepare_Deep_Stop_Mode_Set_CPUCLK</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_STBC&gt;.h</code></td>
<td>R_&lt;Config_STBC&gt;_Check_Module_XXXX_Idle_State</td>
</tr>
<tr>
<td>UART Interface</td>
<td><code>&lt;Config_UARTn&gt;.c</code></td>
<td>R_&lt;Config_UARTn&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_UARTn&gt;_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_UARTn&gt;_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_UARTn&gt;_Receive</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_UARTn&gt;_user.c</code></td>
<td>R_&lt;Config_UARTn&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_UARTn&gt;._interrupt_send</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_UARTn&gt;._interrupt_receive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_UARTn&gt;._interrupt_error</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_UARTn&gt;.h</code></td>
<td>--</td>
</tr>
<tr>
<td>Watchdog Timer</td>
<td><code>&lt;Config_WDTn&gt;.c</code></td>
<td>R_&lt;Config_WDTn&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_WDTn&gt;_Restart</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_WDTXn&gt;.c</code></td>
<td>R_&lt;Config_WDTXn&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_WDTXn&gt;_Restart</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_WDTn&gt;_user.c</code></td>
<td>R_&lt;Config_WDTn&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_WDTn&gt;._interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_WDTXn&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_WDTXn&gt;._interrupt</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_WDTXn&gt;_user.c</code></td>
<td>R_&lt;Config_WDTXn&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_WDTXn&gt;._interrupt</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_WDTn&gt;.h</code></td>
<td>--</td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>File Name</td>
<td>API Function Name</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td><strong>Clock Divide</strong></td>
<td><code>&lt;Config_TAUxn&gt;.c</code></td>
<td><code>R_&lt;Config_TAUxn&gt;_Create</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_TAUxn_m&gt;_Start</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_TAUxn_m&gt;_Stop</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAUxn&gt;_user.c</code></td>
<td><code>R_&lt;Config_TAUxn&gt;_Create_UserInit</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>r_&lt;Config_TAUxn_m&gt;_interrupt</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>r_&lt;Config_TAUxn_m&gt;_interrupt_pek</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAUxn&gt;.h</code></td>
<td>—</td>
</tr>
<tr>
<td><strong>Data CRC</strong></td>
<td><code>&lt;Config_DCRA&lt;n&gt;.c</code></td>
<td><code>R_&lt;Config_DCRA&lt;n&gt;_Create</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_KCRC&lt;n&gt;.c</code></td>
<td><code>R_&lt;Config_DCRA&lt;n&gt;_InitializeCRCData</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_DCRA&lt;n&gt;_Input32bitData</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_DCRA&lt;n&gt;_Input16bitData</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_DCRA&lt;n&gt;_Input8bitData</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_DCRA&lt;n&gt;_GetResult_32bitData</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_DCRA&lt;n&gt;_GetResult_16bitData</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_DCRA&lt;n&gt;_user.c</code></td>
<td><code>R_&lt;Config_DCRA&lt;n&gt;_Create_UserInit</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_KCRC&lt;n&gt;_Create_UserInit</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_DCRA&lt;n&gt;.h</code></td>
<td>—</td>
</tr>
<tr>
<td><strong>Delay Count</strong></td>
<td><code>&lt;Config_TAUxn_m&gt;.c</code></td>
<td><code>R_&lt;Config_TAUxn_m&gt;_Create</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_TAUxn_m&gt;_Start</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_TAUxn_m&gt;_Stop</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAUxn_m&gt;_user.c</code></td>
<td><code>R_&lt;Config_TAUxn_m&gt;_Create_UserInit</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>r_&lt;Config_TAUxn_m&gt;_interrupt</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>r_&lt;Config_TAUxn_m&gt;_interrupt_pek</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAUxn_m&gt;.h</code></td>
<td>—</td>
</tr>
</tbody>
</table>
### Table 2.10 Output File List (10/21)

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>File Name</th>
<th>API Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMA Controller</td>
<td><code>&lt;Config_DMAcn&gt;.c</code></td>
<td>R_ &lt;Config_DMAcn&gt;<em>Create R</em> &lt;Config_DMAcn&gt;<em>Start R</em> &lt;Config_DMAcn&gt;<em>Stop R</em> &lt;Config_DMAcn&gt;_Set_SoftwareTrigger</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_DMAcn&gt;_user.c</code></td>
<td>R_ &lt;Config_DMAcn&gt;<em>Create_UserInit R</em> &lt;Config_DMAcn&gt;<em>dmacnm_interrupt R</em> &lt;Config_DMAcn&gt;_Callback_DMAmn_Transfer_Error</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_DMAcn&gt;.h</code></td>
<td>—</td>
</tr>
<tr>
<td>External Event Count</td>
<td><code>&lt;Config_TAUXn_m&gt;.c</code></td>
<td>R_ &lt;Config_TAUXn_m&gt;<em>Create R</em> &lt;Config_TAUXn_m&gt;<em>Start R</em> &lt;Config_TAUXn_m&gt;_Stop</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAUXn_m&gt;_user.c</code></td>
<td>R_ &lt;Config_TAUXn_m&gt;<em>Create_UserInit R</em> &lt;Config_TAUXn_m&gt;<em>interrupt R</em> &lt;Config_TAUXn_m&gt;_interrupt_pek</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAUXn_m&gt;.h</code></td>
<td>—</td>
</tr>
<tr>
<td>Input Period Count</td>
<td><code>&lt;Config_TAUXn_m&gt;.c</code></td>
<td>R_ &lt;Config_TAUXn_m&gt;<em>Create R</em> &lt;Config_TAUXn_m&gt;<em>Start R</em> &lt;Config_TAUXn_m&gt;<em>Stop R</em> &lt;Config_TAUXn_m&gt;_Get_PulseWidth</td>
</tr>
<tr>
<td>Detection</td>
<td><code>&lt;Config_TAUXn_m&gt;_user.c</code></td>
<td>R_ &lt;Config_TAUXn_m&gt;<em>Create_UserInit R</em> &lt;Config_TAUXn_m&gt;<em>interrupt R</em> &lt;Config_TAUXn_m&gt;_interrupt_pek</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAUXn_m&gt;.h</code></td>
<td>—</td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>File Name</td>
<td>API Function Name</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>----------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Input Position Detection Function</strong></td>
<td><code>&lt;Config_TAUXn_m&gt;.c</code></td>
<td>R_&lt;Config_TAUXn_m&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TAUXn_m&gt;_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TAUXn_m&gt;_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TAUXn_m&gt;_Get_PulseWidth</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAUXn_m&gt;_user.c</code></td>
<td>R_&lt;Config_TAUXn_m&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_TAUXn_m&gt;_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_TAUXn_m&gt;_interrupt_pek</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAUXn_m&gt;.h</code></td>
<td>-</td>
</tr>
<tr>
<td><strong>Input Pulse Interval Judgment</strong></td>
<td><code>&lt;Config_TAUXn_m&gt;.c</code></td>
<td>R_&lt;Config_TAUXn_m&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TAUXn_m&gt;_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TAUXn_m&gt;_Stop</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAUXn_m&gt;_user.c</code></td>
<td>R_&lt;Config_TAUXn_m&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_TAUXn_m&gt;_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_TAUXn_m&gt;_interrupt_pek</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAUXn_m&gt;.h</code></td>
<td>-</td>
</tr>
<tr>
<td><strong>Input Signal Width Judgment</strong></td>
<td><code>&lt;Config_TAUXn_m&gt;.c</code></td>
<td>R_&lt;Config_TAUXn_m&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TAUXn_m&gt;_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TAUXn_m&gt;_Stop</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAUXn_m&gt;_user.c</code></td>
<td>R_&lt;Config_TAUXn_m&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_TAUXn_m&gt;_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_TAUXn_m&gt;_interrupt_pek</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAUXn_m&gt;.h</code></td>
<td>-</td>
</tr>
<tr>
<td><strong>Input Signal Width Measurement</strong></td>
<td><code>&lt;Config_TAUXn_m&gt;.c</code></td>
<td>R_&lt;Config_TAUXn_m&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TAUXn_m&gt;_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TAUXn_m&gt;_Stop</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAUXn_m&gt;_user.c</code></td>
<td>R_&lt;Config_TAUXn_m&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_TAUXn_m&gt;_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_TAUXn_m&gt;_interrupt_pek</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAUXn_m&gt;.h</code></td>
<td>-</td>
</tr>
<tr>
<td><strong>Key Interrupt Function</strong></td>
<td><code>&lt;Config_KEY&gt;.c</code></td>
<td>R_&lt;Config_KEY&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_KEY&gt;_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_KEY&gt;_Stop</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_KEY&gt;_user.c</code></td>
<td>R_&lt;Config_KEY&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_KEY&gt;_interrupt</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_KEY&gt;.h</code></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>One Pulse Output</td>
<td><code>&lt;Config_TAUXn_m&gt;.c</code></td>
<td><code>R_&lt;Config_TAUXn_m&gt;_Create</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_TAUXn_m&gt;_Start</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_TAUXn_m&gt;_Stop</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAUXn_m&gt;_user.c</code></td>
<td><code>R_&lt;Config_TAUXn_m&gt;_Create_UserInit</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>r_&lt;Config_TAUXn_m&gt;_interrupt</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>r_&lt;Config_TAUXn_m&gt;_interrupt_pek</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAUXn_m&gt;.h</code></td>
<td>--</td>
</tr>
<tr>
<td>One-Shot Pulse Output</td>
<td><code>&lt;Config_TAUXn&gt;.c</code></td>
<td><code>R_&lt;Config_TAUXn&gt;_Create</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_TAUXn&gt;_Start</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_TAUXn&gt;_Stop</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_TAUXn&gt;_SoftwareTriggerOn</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAUXn&gt;_user.c</code></td>
<td><code>R_&lt;Config_TAUXn&gt;_Create_UserInit</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>r_&lt;Config_TAUXn&gt;_channelm_interrupt</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>r_&lt;Config_TAUXn&gt;_channelm_interrupt_pek</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAUXn&gt;.h</code></td>
<td>--</td>
</tr>
<tr>
<td>Overflow Interrupt Output (Input period count detecting)</td>
<td><code>&lt;Config_TAUXn_m&gt;.c</code></td>
<td><code>R_&lt;Config_TAUXn_m&gt;_Create</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_TAUXn_m&gt;_Start</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_TAUXn_m&gt;_Stop</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAUXn_m&gt;_user.c</code></td>
<td><code>R_&lt;Config_TAUXn_m&gt;_Create_UserInit</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>r_&lt;Config_TAUXn_m&gt;_interrupt</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>r_&lt;Config_TAUXn_m&gt;_interrupt_pek</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAUXn_m&gt;.h</code></td>
<td>--</td>
</tr>
<tr>
<td>Overflow Interrupt Output (Width Measurement)</td>
<td><code>&lt;Config_TAUXn_m&gt;.c</code></td>
<td><code>R_&lt;Config_TAUXn_m&gt;_Create</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_TAUXn_m&gt;_Start</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_TAUXn_m&gt;_Stop</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAUXn_m&gt;_user.c</code></td>
<td><code>R_&lt;Config_TAUXn_m&gt;_Create_UserInit</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>r_&lt;Config_TAUXn_m&gt;_interrupt</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>r_&lt;Config_TAUXn_m&gt;_interrupt_pek</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TAUXn_m&gt;.h</code></td>
<td>--</td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>File Name</td>
<td>API Function Name</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td><strong>IIC Master Mode</strong></td>
<td><code>&lt;Config_RIICn&gt;.c</code></td>
<td>_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_Master_Send</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_Master_Send_Without_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_Master_Receive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_StartCondition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_StopCondition</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_RIICn&gt;._user.c</code></td>
<td>_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_error_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_receive_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_transmit_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_transmitend_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_callback_transmitend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_callback_receiveend</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_RIICn&gt;._h</code></td>
<td></td>
</tr>
<tr>
<td><strong>IIC Slave Mode</strong></td>
<td><code>&lt;Config_RIICn&gt;.c</code></td>
<td>_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_Slave_Send</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_Slave_Receive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_StartCondition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_StopCondition</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_RIICn&gt;._user.c</code></td>
<td>_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_error_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_receive_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_transmitInterrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_transmitendInterrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_callback_transmitend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>_callback_receiveend</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_RIICn&gt;._h</code></td>
<td></td>
</tr>
</tbody>
</table>
### Table 2.14  Output File List (14/21)

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>File Name</th>
<th>API Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCI3 Clock</td>
<td>&lt;Config_SCI3n&gt;.c</td>
<td>R_ &lt;Config_SCI3n&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_SCI3n&gt;_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_SCI3n&gt;_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_SCI3n&gt;_Send</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_SCI3n&gt;_Receive</td>
</tr>
<tr>
<td>SCI3 Asynchronous Mode</td>
<td>&lt;Config_SCI3n&gt;_user.c</td>
<td>R_ &lt;Config_SCI3n&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_ &lt;Config_SCI3n&gt;_interrupt_send</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_ &lt;Config_SCI3n&gt;_interrupt_receive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_ &lt;Config_SCI3n&gt;_interrupt_error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_ &lt;Config_SCI3n&gt;_interrupt_sendend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_ &lt;Config_SCI3n&gt;_callback_sendend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_ &lt;Config_SCI3n&gt;_callback_receiveend</td>
</tr>
<tr>
<td></td>
<td>&lt;Config_SCI3n&gt;.h</td>
<td>-</td>
</tr>
<tr>
<td>SCI3 Asynchronous Mode</td>
<td>&lt;Config_SCI3n&gt;.c</td>
<td>R_ &lt;Config_SCI3n&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_SCI3n&gt;_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_SCI3n&gt;_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_SCI3n&gt;_Send</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_SCI3n&gt;_Receive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_SCI3n&gt;_Multiprocessor_Send</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_SCI3n&gt;_Multiprocessor_Receive</td>
</tr>
<tr>
<td>SCI3 Asynchronous Mode</td>
<td>&lt;Config_SCI3n&gt;_user.c</td>
<td>R_ &lt;Config_SCI3n&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_ &lt;Config_SCI3n&gt;_interrupt_send</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_ &lt;Config_SCI3n&gt;_interrupt_receive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_ &lt;Config_SCI3n&gt;_interrupt_error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_ &lt;Config_SCI3n&gt;_interrupt_sendend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_ &lt;Config_SCI3n&gt;_callback_sendend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_ &lt;Config_SCI3n&gt;_callback_receiveend</td>
</tr>
<tr>
<td></td>
<td>&lt;Config_SCI3n&gt;.h</td>
<td>-</td>
</tr>
</tbody>
</table>
### Table 2.15 Output File List (15/21)

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>File Name</th>
<th>API Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSPI Master</td>
<td><code>&lt;Config_MSPInm&gt;.c</code></td>
<td><code>R_&lt;Config_MSPInm&gt;_Create</code> &lt;br&gt;<code>R_&lt;Config_MSPInm&gt;_Start</code> &lt;br&gt;<code>R_&lt;Config_MSPInm&gt;_Stop</code> &lt;br&gt;<code>R_&lt;Config_MSPInm&gt;_Send</code> &lt;br&gt;<code>R_&lt;Config_MSPInm&gt;_Receive</code> &lt;br&gt;<code>R_&lt;Config_MSPInm&gt;_Software_Trigger</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_MSPInm&gt;_user.c</code></td>
<td><code>R_&lt;Config_MSPInm&gt;_Create_UserInit</code> &lt;br&gt;<code>r_&lt;Config_MSPInm&gt;_channelnm_interrupt_send</code> &lt;br&gt;<code>r_&lt;Config_MSPInm&gt;_channelnm_interrupt_receive</code> &lt;br&gt;<code>r_&lt;Config_MSPInm&gt;_Callback_Interrupt_Send</code> &lt;br&gt;<code>r_&lt;Config_MSPInm&gt;_Callback_Interrupt_Receive</code> &lt;br&gt;<code>r_&lt;Config_MSPInm&gt;_Callback_Interrupt_Frameend</code> &lt;br&gt;<code>r_&lt;Config_MSPInm&gt;_callback_sendend</code> &lt;br&gt;<code>r_&lt;Config_MSPInm&gt;_callback_receiveend</code> &lt;br&gt;<code>r_&lt;Config_MSPInm&gt;_callback_error</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_MSPInm&gt;.h</code></td>
<td><code>--</code></td>
</tr>
<tr>
<td>MSPI Slave</td>
<td><code>&lt;Config_MSPIn&gt;.c</code></td>
<td><code>R_&lt;Config_MSPIn&gt;_Create</code> &lt;br&gt;<code>R_&lt;Config_MSPIn&gt;_Start</code> &lt;br&gt;<code>R_&lt;Config_MSPIn&gt;_Stop</code> &lt;br&gt;<code>R_&lt;Config_MSPIn&gt;_Send</code> &lt;br&gt;<code>R_&lt;Config_MSPIn&gt;_Receive</code> &lt;br&gt;<code>R_&lt;Config_MSPIn&gt;_Software_Trigger</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_MSPIn&gt;_user.c</code></td>
<td><code>R_&lt;Config_MSPIn&gt;_Create_UserInit</code> &lt;br&gt;<code>r_&lt;Config_MSPIn&gt;_interrupt_send</code> &lt;br&gt;<code>r_&lt;Config_MSPIn&gt;_interrupt_receive</code> &lt;br&gt;<code>r_&lt;Config_MSPIn&gt;_interrupt_framend</code> &lt;br&gt;<code>r_&lt;Config_MSPIn&gt;_interrupt_error</code> &lt;br&gt;<code>r_&lt;Config_MSPIn&gt;_channeln0_interrupt_send</code> &lt;br&gt;<code>r_&lt;Config_MSPIn&gt;_channeln0_interrupt_receive</code> &lt;br&gt;<code>r_&lt;Config_MSPIn&gt;_callback_receiveend</code> &lt;br&gt;<code>r_&lt;Config_MSPIn&gt;_callback_sendend</code> &lt;br&gt;<code>r_&lt;Config_MSPIn&gt;_callback_error</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_MSPIn&gt;.h</code></td>
<td><code>--</code></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><code>&lt;Config_RTC&gt;.c</code></td>
<td><code>R_&lt;Config_RTC&gt;_Create</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_RTC&gt;_Start</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_RTC&gt;_Stop</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_RTC&gt;_Set_HourSystem</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_RTC&gt;_Get_CounterBufferValue</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_RTC&gt;_Get_CounterDirectValue</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_RTC&gt;_Set_CounterValue</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_RTC&gt;_Get_SubCounterValue</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_RTC&gt;_Set_ErrorCorrectionValue</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_RTC&gt;_Set_SubCounterCompareValue</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_RTC&gt;_Get_AlarmValue</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_RTC&gt;_Set_AlarmValue</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_RTC&gt;_Set_AlarmOn</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_RTC&gt;_Set_AlarmOff</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_RTC&gt;_Set_ConstPeriodInterruptOn</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_RTC&gt;_Set_ConstPeriodInterruptOff</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_RTC&gt;_Set_1secondInterruptOn</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_RTC&gt;_Set_1secondInterruptOff</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_RTC&gt;_user.c</code></td>
<td><code>R_&lt;Config_RTC&gt;_Create_UserInit</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_RTC&gt;_interrupt_alarm</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_RTC&gt;_interrupt_periodic</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_RTC&gt;_interrupt_1second</code></td>
</tr>
<tr>
<td>Error Control Module</td>
<td><code>&lt;Config_ECM&gt;.c</code></td>
<td><code>R_&lt;Config_ECM&gt;_Create</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_ECM&gt;_Start</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_ECM&gt;_Stop</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_ECM&gt;_Set_Master_Error</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_ECM&gt;_Set_Checker_Error</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_ECM&gt;_Clear_Master_Error</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_ECM&gt;_Clear_Checker_Error</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_ECM&gt;_Delay_Timer_Start</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_ECM&gt;_Delay_Timer_Stop</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_ECM&gt;_Pseudo_XXXX_Start</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>R_&lt;Config_ECM&gt;_Pseudo_XXXX_Stop</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_ECM&gt;_user.c</code></td>
<td><code>R_&lt;Config_ECM&gt;_Create_UserInit</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>r_&lt;Config_ECM&gt;_ecmmi_interrupt_pek</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>r_&lt;Config_ECM&gt;_mi_interrupt_pek</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>r_&lt;Config_ECM&gt;_dclsmi_interrupt_pek</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_ECM&gt;.h</code></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>GTM Clock</td>
<td><code>&lt;Config_GTM_Clock&gt;.c</code></td>
<td>R_ <code>&lt;Config_GTM_Clock&gt;_Create</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_GT M_CCMn_Create</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_GTM_Clock&gt;_user.c</code></td>
<td>R_ <code>&lt;Config_GTM_Clock&gt;_Create_UserInit</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_GTM_Clock&gt;.h</code></td>
<td>--</td>
</tr>
<tr>
<td>Time Base Unit</td>
<td><code>&lt;Config_TBU&gt;.c</code></td>
<td>R_ <code>&lt;Config_TBU&gt;_Create</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TBU&gt;_Start</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TBU&gt;_Stop</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TBU&gt;_user.c</code></td>
<td>R_ <code>&lt;Config_TBU&gt;_Create_UserInit</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TBU&gt;.h</code></td>
<td>--</td>
</tr>
<tr>
<td>TIM Bit Compression Mode</td>
<td><code>&lt;Config_TIMn&gt;.c</code></td>
<td>R_ <code>&lt;Config_TIMn&gt;_Create</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn&gt;_Start</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn&gt;_Stop</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn&gt;_SetDetectedEdge</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn&gt;_Software_Reset</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn&gt;_NEWVAL_TriggerOn</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn&gt;_ECNTOFL_TriggerOn</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn&gt;_CNTOFL_TriggerOn</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn&gt;_GPROFL_TriggerOn</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn&gt;_TODETOFL_TriggerOn</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn&gt;_Channelm_GLITCHDETOFL_TriggerOn</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TIMn&gt;_user.c</code></td>
<td>R_ <code>&lt;Config_TIMn&gt;_Create_UserInit</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_ <code>&lt;Config_TIMn&gt;_share_interrupt</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_ <code>&lt;Config_TIMn&gt;_chm_share_interrupt</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_ <code>&lt;Config_TIMn&gt;_Callback_GT M_Error</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TIMn&gt;.h</code></td>
<td>--</td>
</tr>
<tr>
<td>TIM Gated Periodic Sampling Mode</td>
<td><code>&lt;Config_TIMn_m&gt;.c</code></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_Create</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_Start</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_Stop</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_UpdateelapsedClock</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_Software_Reset</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_NEWVAL_TriggerOn</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_ECNTOFL_TriggerOn</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_CNTOFL_TriggerOn</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_GPROFL_TriggerOn</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_TODETOFL_TriggerOn</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_GLITCHDETOFL_TriggerOn</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TIMn_m&gt;_user.c</code></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_Create_UserInit</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_ <code>&lt;Config_TIMn_m&gt;_Callback_GT M_Error</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_ <code>&lt;Config_TIMn_m&gt;_share_interrupt</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TIMn_m&gt;.h</code></td>
<td>--</td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>File Name</td>
<td>API Function Name</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>----------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td><strong>TIM Input Prescaler Mode</strong></td>
<td>&lt;Config_TIMn_m&gt;.c</td>
<td>R_&lt;Config_TIMn_m&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TIMn_m&gt;_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TIMn_m&gt;_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TIMn_m&gt;_SetDetectedEdge</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TIMn_m&gt;_Software_Reset</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TIMn_m&gt;_NEWVAL_TriggerOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TIMn_m&gt;_ECNTOFL_TriggerOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TIMn_m&gt;_CNTTOFL_TriggerOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TIMn_m&gt;_GPROFL_TriggerOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TIMn_m&gt;_TODETOFL_TriggerOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TIMn_m&gt;_GLITCHDETOFL_TriggerOn</td>
</tr>
<tr>
<td></td>
<td>&lt;Config_TIMn_m&gt;_user.c</td>
<td>R_&lt;Config_TIMn_m&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_TIMn_m&gt;_Callback_GTM_Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_TIMn_m&gt;_share_interrupt</td>
</tr>
<tr>
<td></td>
<td>&lt;Config_TIMn_m&gt;.h</td>
<td></td>
</tr>
<tr>
<td><strong>TIM Input Event Mode</strong></td>
<td>&lt;Config_TIMn_m&gt;.c</td>
<td>R_&lt;Config_TIMn_m&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TIMn_m&gt;_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TIMn_m&gt;_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TIMn_m&gt;_Software_Reset</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TIMn_m&gt;_NEWVAL_TriggerOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TIMn_m&gt;_ECNTOFL_TriggerOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TIMn_m&gt;_CNTTOFL_TriggerOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TIMn_m&gt;_GPROFL_TriggerOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TIMn_m&gt;_TODETOFL_TriggerOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_TIMn_m&gt;_GLITCHDETOFL_TriggerOn</td>
</tr>
<tr>
<td></td>
<td>&lt;Config_TIMn_m&gt;_user.c</td>
<td>R_&lt;Config_TIMn_m&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_TIMn_m&gt;_Callback_GTM_Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_TIMn_m&gt;_share_interrupt</td>
</tr>
<tr>
<td></td>
<td>&lt;Config_TIMn_m&gt;.h</td>
<td></td>
</tr>
</tbody>
</table>
### Table 2.19  Output File List (19/21)

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>File Name</th>
<th>API Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIM Pulse Integration Mode</td>
<td><code>&lt;Config_TIMn_m&gt;.c</code></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_Create</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_Start</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_Stop</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_Software_Reset</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_NEWVAL_TriggerOn</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_ECNTOFL_TriggerOn</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_CNTOFL_TriggerOn</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_GPROFL_TriggerOn</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_TODETOFL_TriggerOn</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_GLITCHDETOFL_TriggerOn</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TIMn_m&gt;_user.c</code></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_Create_UserInit</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>r_ </code>&lt;Config_TIMn_m&gt;_Callback_GTM_Error`</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>r_ </code>&lt;Config_TIMn_m&gt;_share_interrupt`</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TIMn_m&gt;.h</code></td>
<td>—</td>
</tr>
<tr>
<td>TIM PWM Measurement Mode</td>
<td><code>&lt;Config_TIMn_m&gt;.c</code></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_Create</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_Start</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_Stop</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_Software_Reset</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_NEWVAL_TriggerOn</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_ECNTOFL_TriggerOn</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_CNTOFL_TriggerOn</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_GPROFL_TriggerOn</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_TODETOFL_TriggerOn</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_GLITCHDETOFL_TriggerOn</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TIMn_m&gt;_user.c</code></td>
<td>R_ <code>&lt;Config_TIMn_m&gt;_Create_UserInit</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>r_ </code>&lt;Config_TIMn_m&gt;_Callback_GTM_Error`</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>r_ </code>&lt;Config_TIMn_m&gt;_share_interrupt`</td>
</tr>
<tr>
<td></td>
<td><code>&lt;Config_TIMn_m&gt;.h</code></td>
<td>—</td>
</tr>
</tbody>
</table>
Table 2.20 Output File List (20/21)

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>File Name</th>
<th>API Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIM Serial Shift Mode</td>
<td>&lt;Config_TIMn_m&gt;.c</td>
<td>R_ &lt;Config_TIMn_m&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_TIMn_m&gt;_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_TIMn_m&gt;_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_TIMn_m&gt;_UpdateElapsedClock</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_TIMn_m&gt;_UpdateShiftClock</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_TIMn_m&gt;_UpdateCaptureSource</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_TIMn_m&gt;_UpdateTssmOut</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_TIMn_m&gt;_SetGPR0AsShadowRegister</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_TIMn_m&gt;_Software_Reset</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_TIMn_m&gt;_NEWVAL_TriggerOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_TIMn_m&gt;_ECNTOFL_TriggerOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_TIMn_m&gt;_CNTOFL_TriggerOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_TIMn_m&gt;_GPROFL_TriggerOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_TIMn_m&gt;_TODETOFL_TriggerOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_TIMn_m&gt;_GLITCHDETOFL_TriggerOn</td>
</tr>
<tr>
<td></td>
<td>&lt;Config_TIMn_m&gt;_user.c</td>
<td>R_ &lt;Config_TIMn_m&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_ &lt;Config_TIMn_m&gt;_Callback_GTM_Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_ &lt;Config_TIMn_m&gt;_share_interrupt</td>
</tr>
<tr>
<td></td>
<td>&lt;Config_TIMn_m&gt;.h</td>
<td>---</td>
</tr>
<tr>
<td>ATOM Signal Output Mode Compare</td>
<td>&lt;Config_ATOMn_m&gt;.c</td>
<td>R_ &lt;Config_ATOMn_m&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_ATOMn_m&gt;_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_ATOMn_m&gt;_Stop</td>
</tr>
<tr>
<td></td>
<td>&lt;Config_ATOMn_m&gt;_user.c</td>
<td>R_ &lt;Config_ATOMn_m&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_ &lt;Config_ATOMn_m&gt;_Callback_Shared_IRQk</td>
</tr>
<tr>
<td></td>
<td>&lt;Config_ATOMn_m&gt;.h</td>
<td>---</td>
</tr>
<tr>
<td>ATOM Signal Output Mode Immediate</td>
<td>&lt;Config_ATOMn_m&gt;.c</td>
<td>R_ &lt;Config_ATOMn_m&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_ATOMn_m&gt;_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_ATOMn_m&gt;_Stop</td>
</tr>
<tr>
<td></td>
<td>&lt;Config_ATOMn_m&gt;_user.c</td>
<td>R_ &lt;Config_ATOMn_m&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td>&lt;Config_ATOMn_m&gt;.h</td>
<td>---</td>
</tr>
<tr>
<td>ATOM Signal Output Mode PWM</td>
<td>&lt;Config_ATOMn_m&gt;.c</td>
<td>R_ &lt;Config_ATOMn_m&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_ATOMn_m&gt;_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_ &lt;Config_ATOMn_m&gt;_Stop</td>
</tr>
<tr>
<td></td>
<td>&lt;Config_ATOMn_m&gt;_user.c</td>
<td>R_ &lt;Config_ATOMn_m&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_ &lt;Config_ATOMn_m&gt;_Callback_Shared_IRQk</td>
</tr>
<tr>
<td></td>
<td>&lt;Config_ATOMn_m&gt;.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>ATOM Signal Output Mode Serial</td>
<td>&lt;Config_ATOMn_m&gt;.c</td>
<td>R_&lt;Config_ATOMn_m&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_ATOMn_m&gt;_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_ATOMn_m&gt;_Stop</td>
</tr>
<tr>
<td></td>
<td>&lt;Config_ATOMn_m&gt;_user.c</td>
<td>R_&lt;Config_ATOMn_m&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_ATOMn_m&gt;_Callback_Shared_IRQk</td>
</tr>
<tr>
<td></td>
<td>&lt;Config_ATOMn_m&gt;.h</td>
<td>—</td>
</tr>
<tr>
<td>Dead Time Module</td>
<td>&lt;Config_CDTMn_m&gt;.c</td>
<td>R_&lt;Config_CDTMn_m&gt;_Create</td>
</tr>
<tr>
<td></td>
<td>&lt;Config_CDTMn_m&gt;_user.c</td>
<td>R_&lt;Config_CDTMn_m&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td>&lt;Config_CDTMn_m&gt;.h</td>
<td>—</td>
</tr>
<tr>
<td>DTS Controller</td>
<td>&lt;Config_DTSn&gt;.c</td>
<td>R_&lt;Config_DTSn&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_DTSn&gt;_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_DTSn&gt;_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_DTSn&gt;_Set_SoftwareTrigger</td>
</tr>
<tr>
<td></td>
<td>&lt;Config_DTSn&gt;_user.c</td>
<td>R_&lt;Config_DTSn&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_DTSn&gt;_Callback_Dts_Transfer_End</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_DTSn&gt;_Callback_Dts_Transfer_Count_Match</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_DTSn&gt;_Callback_PEk_Transfer_Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_DTSn&gt;_Callback_Transfer_Error</td>
</tr>
<tr>
<td></td>
<td>&lt;Config_DTSn&gt;.h</td>
<td>—</td>
</tr>
<tr>
<td>ADC Boundary Flag Generator</td>
<td>&lt;Config_ABFG&gt;.c</td>
<td>R_&lt;Config_ABFG&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_ABFG&gt;_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R_&lt;Config_ABFG&gt;_Stop</td>
</tr>
<tr>
<td></td>
<td>&lt;Config_ABFG&gt;_user.c</td>
<td>R_&lt;Config_ABFG&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r_&lt;Config_ABFG&gt;_boundary_flag_pulse_w_interrupt</td>
</tr>
<tr>
<td></td>
<td>&lt;Config_ABFG&gt;.h</td>
<td>—</td>
</tr>
</tbody>
</table>
3. API FUNCTIONS

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
(5) [Return value]

API function return value is explained in the following format.

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

(a) Macro
Macro of return value

(b) Description
Description of return value
### 3.2.1 General

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>main</strong></td>
<td>This is a main function</td>
</tr>
<tr>
<td><strong>R_MAIN_UserInit</strong></td>
<td>Performs user-defined initialization</td>
</tr>
<tr>
<td><strong>R_Systeminit</strong></td>
<td>Performs initialization necessary to control the various peripheral functions</td>
</tr>
<tr>
<td><strong>R_CGC_Create</strong></td>
<td>Performs initialization required to control the clock generator (include reset function, on-chip debug function, etc.)</td>
</tr>
<tr>
<td><strong>R_CGC_Create_UserInit</strong></td>
<td>Performs user-defined initialization relating to the clock generator (Include reset function, on-chip debug function, etc.)</td>
</tr>
<tr>
<td><strong>R_Interrupt_Create</strong></td>
<td>(Reserved function)</td>
</tr>
<tr>
<td><strong>R_Pins_Create</strong></td>
<td>This is a reference for setting the pin function</td>
</tr>
<tr>
<td><strong>R_Interrupt_Initialize_ForPE</strong></td>
<td>Initializes all interrupt of all used modules for PEk</td>
</tr>
<tr>
<td><strong>R_ADC_SyncStart</strong></td>
<td>Simultaneously starts A/D conversion for each scan group of all ADC units</td>
</tr>
<tr>
<td><strong>R_ADC_TimerSyncStart</strong></td>
<td>Simultaneously start A/D timer count operation of all ADC units</td>
</tr>
<tr>
<td><strong>R_ATOMn_Start_Interrupt_IRQ</strong></td>
<td>Starts INTGTMA0ATOMnk interrupt</td>
</tr>
<tr>
<td><strong>R_ATOMn_Stop_Interrupt_IRQ</strong></td>
<td>Stops INTGTMA0ATOMnk interrupt</td>
</tr>
<tr>
<td><strong>r_atomn_irqk_interrupt</strong></td>
<td>Performs processing in response to the INTGTMA0ATOMnk interrupt</td>
</tr>
<tr>
<td><strong>R_SDMAC_Start_Error_Interrupt_PEk</strong></td>
<td>Starts DMAC address error interrupt for PEk</td>
</tr>
<tr>
<td><strong>R_SDMAC_Stop_Error_Interrupt_PEk</strong></td>
<td>Stops DMAC address error interrupt for PEk</td>
</tr>
<tr>
<td><strong>R_DMA_Suspend</strong></td>
<td>Suspends DMA transfer for all channels</td>
</tr>
<tr>
<td><strong>R_DMA_Resume</strong></td>
<td>Resumes DMA transfer for all channels</td>
</tr>
<tr>
<td><strong>R_PDMAn_Suspend</strong></td>
<td>Suspends DMA transfer for all channels</td>
</tr>
<tr>
<td><strong>R_PDMAn_Resume</strong></td>
<td>Resumes DMA transfer for all channels</td>
</tr>
<tr>
<td><strong>R_DMACH_Create</strong></td>
<td>Initialize DMAC module</td>
</tr>
<tr>
<td><strong>R_DMA_Start_Transfer_Error_Interrupt</strong></td>
<td>Starts DMA transfer error interrupt in all DMAC and DTS channels</td>
</tr>
<tr>
<td><strong>R_DMA_Stop_Transfer_Error_Interrupt</strong></td>
<td>Stops DMA transfer error interrupt in all DMAC and DTS channels</td>
</tr>
<tr>
<td><strong>r_sdmac_address_error_interrupt_pek</strong></td>
<td>Performs processing in response to the SDMAC address error interrupt for Pek</td>
</tr>
<tr>
<td><strong>r_dmac_error_interrupt_pek</strong></td>
<td>Performs processing in response to the DMA transfer error interrupt for PE1</td>
</tr>
<tr>
<td><strong>R_DTS_Suspend</strong></td>
<td>Suspends DTS transfer</td>
</tr>
<tr>
<td><strong>R_DTS_Resume</strong></td>
<td>Resumes DTS transfer</td>
</tr>
<tr>
<td><strong>R_DTSg_Start_Transfer_EndInterrupt</strong></td>
<td>Clears DTS channel g transfer end interrupt request and enable operation</td>
</tr>
<tr>
<td><strong>R_DTSg_Stop_Transfer_EndInterrupt</strong></td>
<td>Disables DTS channel g transfer end interrupt operation and clear request</td>
</tr>
</tbody>
</table>
Table 3.1.2 API Functions: [General] (2/2)

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_DTSg_Start_Transfer_Count_Match_Interrupt</td>
<td>Clears DTS channel $g$ transfer count match interrupt request and enable operation</td>
</tr>
<tr>
<td>R_DTSg_Stop_Transfer_Count_Match_Interrupt</td>
<td>Disables DTS channel $g$ transfer count match interrupt operation and clear request</td>
</tr>
<tr>
<td>R_DTS_Start_PEk_Transfer_Error_Interrupt</td>
<td>Clears DTS transfer error interrupt request and enable operation for PEk</td>
</tr>
<tr>
<td>R_DTS_Stop_PEk_Transfer_Error_Interrupt</td>
<td>Disables DTC transfer error interrupt operation and clear request for PEk</td>
</tr>
<tr>
<td>r_dtsg_transfer_end_interrupt</td>
<td>Performs processing in response to the DTS channel $g$ transfer end interrupt</td>
</tr>
<tr>
<td>r_dtsg_transfer_count_match_interrupt</td>
<td>Performs processing in response to the DTS channel $g$ transfer count match interrupt</td>
</tr>
<tr>
<td>r_dts_transfer_error_interrupt_pek</td>
<td>Performs processing in response to the DTS transfer error interrupt for PEk</td>
</tr>
<tr>
<td>R_GTM_Start_Interrupt_Error</td>
<td>Enables GTM error interrupt</td>
</tr>
<tr>
<td>R_GTM_Stop_Interrupt_Error</td>
<td>Disables GTM error interrupt</td>
</tr>
<tr>
<td>r_gtm_error_interrupt</td>
<td>Performs processing in response to the GTM error interrupt</td>
</tr>
<tr>
<td>R_MSPI_Master_Create</td>
<td>Performs initialization for all MSPI channels</td>
</tr>
<tr>
<td>R_MSPI_Start_Interrupt_MSPIinTX</td>
<td>Enables MSPI$n$ transmit interrupt</td>
</tr>
<tr>
<td>R_MSPI_Stop_Interrupt_MSPIinTX</td>
<td>Disables MSPI$n$ transmit interrupt</td>
</tr>
<tr>
<td>R_MSPI_Start_Interrupt_MSPIinRX</td>
<td>Enables MSPI$n$ receive interrupt</td>
</tr>
<tr>
<td>R_MSPI_Stop_Interrupt_MSPIinRX</td>
<td>Disables MSPI$n$ receive interrupt</td>
</tr>
<tr>
<td>R_MSPI_Start_Interrupt_MSPInFE</td>
<td>Enables MSPI$n$ frame end interrupt</td>
</tr>
<tr>
<td>R_MSPI_Stop_Interrupt_MSPInFE</td>
<td>Disables MSPI$n$ frame end interrupt</td>
</tr>
<tr>
<td>R_MSPI_Start_Interrupt_MSPInERR</td>
<td>Enables MSPI$n$ error interrupt</td>
</tr>
<tr>
<td>R_MSPI_Stop_Interrupt_MSPInERR</td>
<td>Disables MSPI$n$ error interrupt</td>
</tr>
<tr>
<td>r_mspin_interrupt_send</td>
<td>Performs processing in response to the MSPI Master communication interrupt</td>
</tr>
<tr>
<td>r_mspin_interrupt_receive</td>
<td>Performs processing in response to the MSPI$n$ reception interrupt</td>
</tr>
<tr>
<td>r_mspin_interrupt_frameend</td>
<td>Performs processing in response to the MSPI$n$ frame end interrupt</td>
</tr>
<tr>
<td>r_mspin_interrupt_error</td>
<td>Performs processing in response to the MSPI$n$ error interrupt</td>
</tr>
</tbody>
</table>
This is the main function.

[Syntax]
void main(void);

[Argument(s)]
None.

[Return value]
None.
R_MAIN_UserInit

Performs user-defined initialization.

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
**R_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 by R_Systeminit function.

[Syntax]
void R_CGC_Create(void);

[Argument(s)]
None.

[Return value]
None.
**R_CGC_Create_UserInit**

Performs user-defined initialization relating to the clock generator (include reset function, on-chip debug function, etc.).

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

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

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

**[Return value]**
None.
R_Interrupt_Initialize_ForPE

Performs initialization of all interrupts for each CPU core.

Remark This API function is called by R_Systeminit.

[Syntax]

```c
void R_Interrupt_Initialize_ForPE(void);
```

[Argument(s)]

None.

[Return value]

None.
R_ADC_SyncStart

Starts A/D conversion for each scan group of all A/D Converter units simultaneously.

Remark  This API function is called by user.

[Syntax]

```c
void R_ADC_SyncStart(void);
```

[Argument(s)]

None.

[Return value]

None.
R_ADC_TimerSyncStart

Starts A/D timer count operation of all A/D Converter units simultaneously.

Remark    This API function is called by user.

[Syntax]

```c
void R_ADC_TimerSyncStart(void);
```

[Argument(s)]

None.

[Return value]

None.
R_DMAC_Create

Initializes all DMAC channels

Remark  This API function is called by R_Systeminit.

[Syntax]

void    R_DMAC_Create(void)

[Argument(s)]

None.

[Return value]

None.
**R_DMA_Suspend**

Suspend DMA transfer for all channels.

**Remark**  
This API function is called by user.

**[Syntax]**

```c
void R_DMA_Suspend(void)
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_DMA_Resume

Resumes DMA transfer for all channels

Remark  This API function is called by user.

[Syntax]

void  R_DMA_Resume(void)

[Argument(s)]

None.

[Return value]

None.
R_PDMA\_n\_Suspend

Suspends DMA transfer for all channels of unit \( n \).

Remark  This API function is called by user.

[Syntax]

\[
\text{void R_PDMA}\_n\_Suspend(\text{void})
\]

Remark  \( n \) is the unit number.

[Argument(s)]

None.

[Return value]

None.
R_PDMA\textsubscript{n} Resume

Resumes DMA transfer for all channels of unit \textit{n}.

Remark This API function is called by user.

[Syntax]

\begin{verbatim}
void R_PDMA\textsubscript{n} Resume(void)
\end{verbatim}

Remark \textit{n} is the unit number.

[Argument(s)]

None.

[Return value]

None.
**R_DMAC_Start_Error_Interrupt_PEk**

Starts SDMAC address error interrupt for PEk.

**Remark** This API function is called by user.

**[Syntax]**

```c
void R_DMAC_Start_Error_Interrupt_PEk(void);
```

**Remark** $k$ is the CPU core number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_DMAC_Stop_Error_Interrupt_PEk

Stops SDMAC address error interrupt for PEk.

Remark This API function is called by user.

[Syntax]

void R_DMAC_Stop_Error_Interrupt_PEk(void);

Remark  k is the CPU core number.

[Argument(s)]

None.

[Return value]

None.
R_DMA_Start_Transfer_Error_Interrupt

Starts DMA transfer error interrupt for all DMAC and DTS channels.

Remark    This API function is called by user.

[Syntax]

```c
void R_DMA_Start_Transfer_Error_Interrupt(void);
```

[Argument(s)]

None.

[Return value]

None.
R_DMA_Stop_Transfer_Error_Interrupt

Stops DMA transfer error interrupt for all DMAC and DTS channels.

Remark   This API function is called by user.

[Syntax]

void    R_DMA_Stop_Transfer_Error_Interrupt(void);

[Argument(s)]

None.

[Return value]

None.
**r_sdmac_address_error_interrupt_pek**

Performs processing in response to the SDMAC address error interrupt.

**[Syntax]**

```c
void r_sdmac_address_error_interrupt_pek(void);
```

Remark: $k$ is the CPU core number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_dmac_error_interrupt_pek

Performs processing in response to the DMA transfer error interrupt for PEk

[Syntax]

```c
void r_dmac_error_interrupt_pek(void);
```

Remark $k$ is the CPU core number (RH850/C1M supports PE1 only).

[Argument(s)]
None.

[Return value]
None.
R_DTS_Create

Initializes DTS module.

Remark This API function is called by R_Systeminit.

[Syntax]

```
void R_DTS_Create(void)
```

[Argument(s)]

None.

[Return value]

None.
### R_DTS_Suspend

Suspend DTS transfer.

**Remark**

This API function is called by user.

**[Syntax]**

```c
void R_DTS_Suspend(void)
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_DTS_Resume

Resumes DTS transfer.

Remark  This API function is called by user.

[Syntax]

```c
void R_DTS_Resume(void)
```

[Argument(s)]

None.

[Return value]

None.
R_DTSg_Start_Transfer_End_Interrupt

Starts DTS channel \( g \) transfer end interrupt.

Remark  This API function is called by user.

[Syntax]

```c
void R_DTSg_Start_Transfer_End_Interrupt(void)
```

Remark  \( g \) is interrupt group number, 31_0, 63_32, 95_64, 127_96.

[Argument(s)]

None.

[Return value]

None.
R_DTSg_Stop_Transfer_End_Interrupt

Stops DTS channel \( g \) transfer end interrupt.

Remark This API function is called by user.

[Syntax]

```c
void R_DTSg_Stop_Transfer_End INTERRUPTION(void)
```

Remark \( g \) is interrupt group number, 31_0, 63_32, 95_64, 127_96.

[Argument(s)]

None.

[Return value]

None.
R_DTSg_Start_Transfer_Count_Match_Interrupt

Starts DTS channel $g$ transfer count match interrupt.
Remark This API function is called by user.

[Syntax]

```c
void R_DTSg_Start_Transfer_Count_Match_Interrupt(void)
```

Remark $g$ is interrupt group number, 31_0, 63_32, 95_64, 127_96.

[Argument(s)]

None.

[Return value]

None.
R_DTS\textsubscript{g} Stop Transfer Count Match Interrupt

Stops DTS channel \( g \) transfer count match interrupt.

Remark This API function is called by user.

[Syntax]

\begin{verbatim}
void R_DTS\textsubscript{g} Stop Transfer Count Match Interrupt(void)
\end{verbatim}

Remark \( g \) is interrupt group number, 31\_0, 63\_32, 95\_64, 127\_96.

[Argument(s)]

None.

[Return value]

None.
**R_DTS_Start_PEk_Transfer_Error_Interrupt**

Starts DTS transfer error interrupt for PEk.

**Remark**  This API function is called by user.

**[Syntax]**

```c
void R_DTS_Start_PEk_Transfer_Error_Interrupt (void)
```

**Remark**  \( k \) is the CPU core number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_DTS_Stop_PEk_Transfer_Error_Interrupt

Stops DTS transfer error interrupt.
Remark This API function is called by user.

[Syntax]

```c
void R_DTS_Stop_PEk_Transfer_Error_Interrupt(void)
```

Remark $k$ is the CPU core number.

[Argument(s)]
None.

[Return value]
None.
r_dtsg_transfer_end_interrupt

Performs processing in response to the DTS channel g transfer end interrupt.

[Syntax]

```c
void r_dtsg_transfer_end_interrupt (void)
```

Remark  

\( g \) is interrupt group number, 31to0, 63to32, 95to64, 127to96.

[Argument(s)]

None.

[Return value]

None.
[Syntax]

```c
void r_dts_g_transfer_count_match_interrupt (void)
```

Remark  

`g` is interrupt group number, 31to0, 63to32, 95to64, 127to96.

[Argument(s)]

None.

[Return value]

None.
r_dts_transfer_error_interrupt_pek

Performs processing in response to the DTS transfer error interrupt for PEk.

[Syntax]

```c
void r_dts_transfer_error_interrupt_pek (void)
```

Remark  

$k$ is CPU core number.

[Argument(s)]

None.

[Return value]

None.
R_GTM_Start_Interrupt_Error

Enables GTM error interrupt.

Remark  This API functions is called indirectly by R_<Config_TIMn>_Start(TIM Bit Compression Mode), R_<Config_TIMn_m>_Start(TIM Gated Periodic Sampling Mode), R_<Config_TIMn_m>_Start(TIM Input Prescaler Mode), R_<Config_TIMn_m>_Start(TIM input event mode), R_<Config_TIMn_m>_Start(TIM pulse integration mode), R_<Config_TIMn_m>_Start(TIM pwm measurement mode), R_<Config_TIMn_m>_Start(TIM serial shift mode).

[Syntax]

```c
void R_GTM_Start_Interrupt_Error(void);
```

[Argument(s)]

None.

[Return value]

None.
R_GTM_Stop_Interrupt_Error

Disables GTM error interrupt.

Remark  This API function is called indirectly by R_<Config_TIMn>_Stop (TIM Bit Compression Mode), R_<Config_TIMn_m>_Stop (TIM Gated Periodic Sampling Mode), R_<Config_TIMn_m>_Stop (TIM Input Prescaler Mode), R_<Config_TIMn_m>_Stop (TIM input event mode), R_<Config_TIMn>_Stop (TIM pulse integration mode), R_<Config_TIMn_m>_Stop (TIM pwm measurement mode), R_<Config_TIMn_m>_Stop (TIM serial shift mode).

[Syntax]

```c
void R_GTM_Stop_Interrupt_Error(void);
```

[Argument(s)]

None.

[Return value]

None.
r_gtm_error_interrupt

Performs processing in response to the GTM error interrupt.

[Syntax]

```c
void r_gtm_error_interrupt(void);
```

[Argument(s)]

None.

[Return value]

None.
R_ATOMn_Start_Interrupt_IRQk

Starts INTGTMA0ATOMnk interrupt.

Remark  This API function is called indirectly by R_<Config_ATOMn_m>_Start(ATOM Signal Output Mode Compare), R_<Config_ATOMn_m>_Start(ATOM Signal Output Mode PWM), R_<Config_ATOMn_m>_Start(ATOM Signal Output Mode Serial) function.

[Syntax]

```c
void R_ATOMn_Start_Interrupt_IRQk(void);
```

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

[Argument(s)]

None.

[Return value]

None.
R_ATOMn_Stop_Interrupt_IRQk

Stops INTGTMATOMnk interrupt.

Remark This API function is called indirectly by R_<Config_ATOMn_m>_Stop (ATOM Signal Output Mode Compare), R_<Config_ATOMn_m>_Stop (ATOM Signal Output Mode PWM), R_<Config_ATOMn_m>_Stop (ATOM Signal Output Mode Serial) function.

[Syntax]

void R_ATOMn_Stop_Interrupt_IRQk(void);

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

[Argument(s)]

None.

[Return value]

None.
r_atomn_irqk_interrupt

Performs processing in response to the INTGMA0ATOMnk interrupt.

[Syntax]

```c
void r_atomn_irqk_interrupt(void);
```

Remark  

\( n \) is the unit number, \( k \) is the interrupt channel number.

[Argument(s)]

None.

[Return value]

None.
### R_MSPI_Master_Create

Performs initialization for all MSPI channels.

**Remark** This API function is called by `R_Systeminit` function.

#### [Syntax]

```c
void R_MSPI_Master_Create(void);
```

#### [Argument(s)]

None.

#### [Return value]

None.
R_MSPI_Start_Interrupt_MSPI\textsubscript{n}TX

Enables MSPI\textsubscript{n} transmit interrupt

Remark  This API functions is called by user.

[Syntax]

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

Remark  \( n \) is the unit number.

[Argument(s)]

None.

[Return value]

None.
R_MSPI_Stop_Interrupt_MSPI\textsubscript{nTX}

Disables \text{MSPI\textsubscript{n}} transmit interrupt

Remark This API function is called by user.

**Syntax**

```c
void R_MSPI_Stop_Interrupt_MSPI\textsubscript{nTX}(void);
```

Remark \text{n} is the unit number.

**Argument(s)**

None.

**Return value**

None.
R_MSPI_Start_Interrupt_MSPI\textsubscript{n}RX

Enables MSPI\textsubscript{n} receive interrupt

Remark This API functions is called by user.

[Syntax]

void R_MSPI\_Start\_Interrupt\_MSPI\textsubscript{n}RX(void);

Remark \( n \) is the unit number.

[Argument(s)]

None.

[Return value]

None.
R_MSPI_Stop_Interrupt_MSPI\textsubscript{nRX}

Disables MSPI\textsubscript{n} receive interrupt

Remark This API function is called by user.

[Syntax]

\begin{verbatim}
void R_MSPI_Stop_Interrupt_MSPI\textsubscript{nRX}(void);
\end{verbatim}

Remark \( n \) is the unit number.

[Argument(s)]

None.

[Return value]

None.
R_MSPI_Start_Interrupt_MSPI\textsubscript{n}FE

Enables MSPI\textsubscript{n} frame end interrupt

Remark  This API functions is called by user.

[Syntax]

\begin{verbatim}
void R_MSPI_Start_Interrupt_MSPI\textsubscript{n}FE(void);
\end{verbatim}

Remark  \( n \) is the unit number.

[Argument(s)]

None.

[Return value]

None.
**R_MSPI_Stop_Interrupt_MSPA FE**

Disables MSPIn frame end interrupt

Remark This API functions is called by user.

**[Syntax]**

```c
void R_MSPI_Stop_Interrupt_MSPA FE(void);
```

Remark \( n \) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_MSPI_Start_Interrupt_MSPI\textsubscript{\textit{n}}\textsubscript{\textit{ERR}}

Enables MSPI\textsubscript{\textit{n}} error interrupt

Remark This API functions is called by user.

[Syntax]

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

Remark \textit{n} is the unit number.

[Argument(s)]

None.

[Return value]

None.
R_MSPI_Stop_Interrupt_MSPI\textsubscript{n}ERR

Disables MSPI\textsubscript{n} error interrupt

Remark This API function is called by user.

[Syntax]

\begin{verbatim}
void R_MSPI_Stop_Interrupt_MSPI\textsubscript{n}ERR(void);
\end{verbatim}

Remark \( n \) is the unit number.

[Argument(s)]

None.

[Return value]

None.
r_mspin_interrupt_send

Performs processing in response to the MSPI Master communication interrupt.

**[Syntax]**

```
void r_mspin_interrupt_send(void);
```

Remark $n$ is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_mspi_interrupt_receive

Performs processing in response to the MSPI{n} reception interrupt.

[Syntax]

```c
void r_mspi_interrupt_receive(void);
```

Remark  \( n \) is the unit number.

[Argument(s)]

None.

[Return value]

None.
r_mspin_interrupt_frameend

Performs processing in response to the MSPI\textsubscript{n} frameend interrupt.

**[Syntax]**

```c
void r_mspin_interrupt_frameend(void);
```

Remark \textit{n} is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_mspin_interrupt_error**

Performs processing in response to the MSPI\textit{n} error interrupt.

**[Syntax]**

```c
void r_mspin_interrupt_error(void);
```

Remark \textit{n} is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R Interrupt Create**

Reserved function (Processing code is not output).

**[Syntax]**

```c
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 by any function.

[Syntax]

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

[Argument(s)]
None.

[Return value]
None.
### 3.2.2 A/D Converter

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

**Table 3.2.1 API Functions: [A/D Converter] (1/2)**

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_ADCXn&gt;_Create</td>
<td>Performs initialization necessary to control the group scan mode</td>
</tr>
<tr>
<td></td>
<td>A/D Converter functions</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_Halt</td>
<td>Halts A/D converter</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_SelfDiagnostic_VoltageCircuitOn</td>
<td>Turns on the self-diagnostic voltage circuit or updates the reference voltage.</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_SelfDiagnostic_VoltageCircuitOff</td>
<td>Turns off the self-diagnostic voltage circuit.</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_ScanGroupx_Start</td>
<td>Starts A/D converter of scan group</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_ScanGroupx_OperationOn</td>
<td>Starts scan group scan</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_ScanGroupx_OperationOff</td>
<td>Stops scan group scan</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_ScanGroupx_GetResult</td>
<td>Reads the A/D conversion results of scan group</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_ScanGroupx_GetPWMDiagResult</td>
<td>Gets A/D conversion results for ADC scan group (PWM-Diag).</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_ScanGroupx_GetFloatingPointResult</td>
<td>Gets A/D conversion result in floating point format for ADC scan group</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_SetMultiplexerCommand</td>
<td>Sets multiplexer command for ADC.</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_StrongPullDown_Start</td>
<td>Enables strong pull-down of the target physical channel.</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_StrongPullDown_Stop</td>
<td>Disables strong pull-down of the target physical channel.</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_TH_Groupk_Start</td>
<td>Starts T&amp;H group hold</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_TH_Sampling_Start</td>
<td>Starts sampling to all TH.</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_TH_Stop</td>
<td>Stops all T&amp;H operation</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_SetMultiplexerCommand</td>
<td>Sets multiplexer command for ADC.</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_ScanGroupx_TimerStart</td>
<td>Starts timer of ADC scan group</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_ScanGroupx_TimerStop</td>
<td>Stops timer of ADC scan group</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_VoltageDivider_Start</td>
<td>Enables voltage monitoring voltage divider</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_VoltageDivider_Stop</td>
<td>Disables voltage monitoring voltage divider</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_StrongPullDown_Start</td>
<td>Enables strong pulldown for physical channel which has strong pulldown</td>
</tr>
<tr>
<td></td>
<td>function</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_StrongPullDown_Stop</td>
<td>Disables strong pulldown for physical channel which has strong pulldown</td>
</tr>
<tr>
<td></td>
<td>function</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_SGDiag_Start</td>
<td>Enables A/D conversion of Diag scan groups</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_SGDiag_OperationOn</td>
<td>Starts A/D conversion of Diag scan groups</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_SGDiag_OperationOff</td>
<td>Stops A/D conversion of Diag scan groups</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_SGDiag_GetResult</td>
<td>Gets A/D conversion results for Diag scan groups</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_SGDiag_GetSubtractionResult</td>
<td>Gets all subtraction results for Diag scan groups</td>
</tr>
</tbody>
</table>
### Table 3.3.2 API Functions: [A/D Converter] (2/2)

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_ADCXn&gt;_ASF_Start</td>
<td>Starts of ASF accumulation counter</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_ASF_Stop</td>
<td>Stops of ASF accumulation counter</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_ASF_GetResult</td>
<td>Gets accumulation result by accumulation channel</td>
</tr>
<tr>
<td>R_&lt;Config_ADCXn&gt;_ASF_GetRealTimeResult</td>
<td>Gets accumulation intermediate(real time) result by accumulation channel</td>
</tr>
<tr>
<td>r_&lt;Config_ADCXn&gt;_error_interrupt</td>
<td>Performs processing in response to the A/D error interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_ADCXn&gt;_scan_groupx_end_interrupt</td>
<td>Performs processing in response to the scan group end interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_ADCXn&gt;_monitor_error_interrupt</td>
<td>Performs processing in response to the monitor error interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_ADCXn&gt;_sg_diag_end_interrupt</td>
<td>Performs processing in response to SG-Diag end interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_ADCXn&gt;_mpx_request_interrupt</td>
<td>Performs processing in response to MPX DMA trigger request interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_ADCXn&gt;_asf_channelk_end_interrupt</td>
<td>Performs processing in response to accumulation end interrupt</td>
</tr>
</tbody>
</table>
R_<Config_ADCXn>_Create

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

Remark This API function is called by R_Systeminit function.

[Syntax]
void R_<Config_ADCXn>_Create(void);

Remark ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit number.

[Argument(s)]
None.

[Return value]
None.
**R_<Config_ADCXn>_Halt**

Halts A/D converter.

**[Syntax]**

```c
void R_<Config_ADCXn>_Halt(void);
```

**Remark**

ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_ADCXn>_SelfDiagnostic_VoltageCircuitOn

Turns on the self-diagnostic voltage circuit or updates the reference voltage.

[Syntax]

```c
void R_<Config_ADCXn>_SelfDiagnostic_VoltageCircuitOn(void);
```

Remark ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit number.

[Argument(s)]

None.

[Return value]

None.
R_<Config_ADCXn>_SelfDiagnostic_VoltageCircuitOff

Turns off the self-diagnostic voltage circuit.

**[Syntax]**

```c
void R_<Config_ADCXn>_SelfDiagnostic_VoltageCircuitOff(void);
```

**Remark**

ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R\_**<Config\_ADC\_Xn>_\_ScanGroup\_x\_Start

Enables A/D converter of scan group.

**[Syntax]**

```c
void R_<Config_ADC_Xn>_ScanGroup_x_Start(void);
```

**Remark**
ADC\_X is ADC name (ADCA, ADCC, ADCJ, ADCK), \( n \) is the unit number, \( x \) is the scan group number.

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

**[Return value]**
None.
**R_<Config_ADCXn>_ScanGroupx(OperationOn**

Starts scan group scan by using a software trigger.

**[Syntax]**

```c
void R_<Config_ADCXn>_ScanGroupx(OperationOn(void);
```

Remark ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit number, x is the scan group number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_ADCXn>_ScanGroupx_OperationOff**

Stops scan group scan by using a software trigger.

**[Syntax]**

```c
void R_<Config_ADCXn>_ScanGroupx_OperationOff(void);
```

**Remark**
ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit number, x is the scan group number.

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

**[Return value]**
None.
R_<Config_ADCXn>_ScanGroupx_GetResult

Reads the A/D conversion results of scan group.

[Syntax]

```c
MD_STATUS R_<Config_ADCXn>_ScanGroupx_GetResult(uint16_t * const buffer, uint8_t buffer_size);
```

Remark: ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), 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>
<tr>
<td>I</td>
<td>uint8_t buffer_size</td>
<td>The size of buffer</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>The actual converted data number is beyond the buffer_size</td>
</tr>
</tbody>
</table>

R_<Config_ADCXn>_ScanGroupx_GetPWMDiagResult

Gets A/D conversion results for ADC scan group (PWM-Diag).

[Syntax]

```c
MD_STATUS R_<Config_ADCXn>_ScanGroupx_GetPWMDiagResult(uint16_t * const buffer);
```

Remark: ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), 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]

<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_ADCXn>_ScanGroupx_GetFloatingPointResult**

Gets A/D conversion result in floating point format for scan group.

**[Syntax]**

\[
\text{MD\_STATUS } \text{R}_{<\text{Config\_ADCXn}>}_\text{ScanGroupx\_GetFloatingPointResult}(\text{uint32\_t\_const buffer, uint8\_t buffer\_size});
\]

Remark 
ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), \(n\) 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>uint32_t_const buffer</td>
<td>Pointer to area in which to store the results of A/D conversion</td>
</tr>
<tr>
<td>I</td>
<td>uint8_t buffer_size</td>
<td>The size of buffer</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>The actual converted data number is beyond the buffer_size</td>
</tr>
</tbody>
</table>
**R\_<Config\_ADCXn>_SetMultiplexerCommand**

Sets multiplexer command for ADC.

**[Syntax]**

```c
void R\_<Config\_ADCXn>_SetMultiplexerCommand;
```

Remark ADCX is ADC name (ADCK), n is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_ADCXn>_TH_Groupk_Start

Starts T&H group hold.

[Syntax]

```c
void R_<Config_ADCXn>_TH_Groupk_Start(void);
```

Remark

ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit number, k is the T&H group number.

[Argument(s)]

None.

[Return value]

None.
R_<Config_ADCXn>_TH_Sampling_Start

Starts T&H sampling.

[Syntax]

void R_<Config_ADCXn>_TH_Sampling_Start(void);

Remark
ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit number.

[Argument(s)]
None.

[Return value]
None.
R_\text{<Config}_\text{ADCX}n\text{_TH_Stop}

Stops Stops all T&H operation.

[Syntax]

\begin{verbatim}
void R_\text{<Config}_\text{ADCX}n\text{_TH_Stop}(void);
\end{verbatim}

Remark ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), \( n \) is the unit number.

[Argument(s)]

None.

[Return value]

None.
R_<Config_ADCXn>_SetMultiplexerCommand

Sets multiplexer command for ADC.

[Syntax]

```c
void R_<Config_ADCXn>_SetMultiplexerCommand(uint8_t cmdData);
```

Remark  
ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit 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 cmdData</td>
<td>Multiplexer command</td>
</tr>
</tbody>
</table>

[Return value]

None.
### R_<_Config_ADCXn>_ScanGroupx_TimerStart

Starts timer of ADC scan group.

#### Syntax

```c
void R_<_Config_ADCXn>_ScanGroupx_TimerStart(void);
```

**Remark**
ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit number, x is the scan group number.

#### Argument(s)
None.

#### Return value
None.
**R_<Config_ADCXn>_ScanGroupx_TimerStop**

Stops timer of ADC scan group.

**[Syntax]**

```c
void R_<Config_ADCXn>_ScanGroupx_TimerStop(void);
```

**Remark**

ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit number, x is the scan group number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_ADCXn>_VoltageDivider_Start

Enables voltage monitoring voltage divider.

[Syntax]

```c
void R_<Config_ADCXn>_VoltageDivider_Start(void);
```

[Remark] ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit number.

[Argument(s)]

None.

[Return value]

None.
Disables voltage monitoring voltage divider.

[Syntax]
```c
void R_<Config_ADCXn>_VoltageDivider_Stop(void);
```

Remark
ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit number.

[Argument(s)]
None.

[Return value]
None.
**R_<Config_ADCXn>_StrongPullDown_Start**

Enables strong pull-down function for physical channel which has strong pull-down function.

**[Syntax]**

```c
void R_<Config_ADCXn>_StrongPullDown_Start(void);
```

Remark ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
Disables strong pull-down function for physical channel which has strong pull-down function.

**[Syntax]**

```c
void R_<Config_ADCXn>_StrongPullDown_Stop(void);
```

**Remark**

ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_ADCXn>_SGDiag_Start

Enables A/D conversion of Diag scan groups

**[Syntax]**

```c
void R_<Config_ADCXn>_SGDiag_Start(void);
```

**Remark**
ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit number.

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

**[Return value]**
None.
R_<Config_ADCXn>_SGDiag_OperationOn

Starts A/D conversion of Diag scan groups by using a software trigger.

**[Syntax]**

```c
void R_<Config_ADCXn>_SGDiag_OperationOn(void);
```

**Remark**

ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_ADCXn>_SGDiag_OperationOff

Stops A/D conversion of Diag scan groups by using a software trigger.

[Syntax]

```c
void R_<Config_ADCXn>_SGDiag_OperationOff(void);
```

[Argument(s)]
None.

[Return value]
None.

Remark ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit number.
R_{<Config_ADCXn>_SGDiag_GetResult}

Gets the A/D conversion results of Diag scan groups.

[Syntax]

```c
MD_STATUS R_{<Config_ADCXn>_SGDiag_GetResult}(uint16_t * const buffer, uint8_t buffer_size);
```

Remark: ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit 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>
<tr>
<td>I</td>
<td>uint8_t buffer_size</td>
<td>The size of buffer</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>The actual converted data number is beyond the buffer_size</td>
</tr>
</tbody>
</table>
R_<Config_ADCXn>_SGDiag_GetSubtractionResult

Gets all subtraction results for Diag scan groups.

[Syntax]

```c
MD_STATUS R_<Config_ADCXn>_SGDiag_GetSubtractionResult(uint32_t * const buffer, uint8_t buffer_size);
```

Remark  ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), \( n \) is the unit 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 buffer</td>
<td>Pointer to area in which to store the results of A/D conversion</td>
</tr>
<tr>
<td>I</td>
<td>uint8_t buffer_size</td>
<td>The size of buffer</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>The actual converted data number is beyond the buffer_size</td>
</tr>
</tbody>
</table>

R_<Config_ADCXn>_ASF_Start

Starts ASF accumulation counter.

**[Syntax]**

```c
void R_<Config_ADCXn>_ASF_Start(void);
```

**Remark**
ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit number.

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

**[Return value]**
None.
R_<Config_ADCXn>_ASF_Stop

Stops ASF accumulation counter.

[Syntax]
void R_<Config_ADCXn>_ASF_Stop(void);

Remark ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit number.

[Argument(s)]
None.

[Return value]
None.
**R_</Config_ADCXn>_ASF_GetResult**

Gets accumulation result by accumulation channel when the count value of the accumulation counter matches with accumulation compare value.

**[Syntax]**

```c
MD_STATUS R_</Config_ADCXn>_ASF_GetResult(uint8_t channel, uint32_t* const data);
```

**Remark**

ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit 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 channel</td>
<td>Accumulation channel i (i=0 to 15)</td>
</tr>
<tr>
<td>O</td>
<td>uint32_t * const data</td>
<td>Data pointer to area in which to store an accumulation data</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>The channel is out of the accumulation channel range (0 to 15)</td>
</tr>
</tbody>
</table>
R_{<Config_{ADCXn}>}.ASF_GetRealTimeResult

Gets accumulation intermediate (real time) result by accumulation channel.

**[Syntax]**

```c
MD_STATUS R_{<Config_{ADCXn}>}.ASF_GetRealTimeResult(uint8_t channel, uint32_t* const data, uint16_t* const counter);
```

**Remark**
ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit 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 channel</td>
<td>Accumulation channel i (i=0 to 15)</td>
</tr>
<tr>
<td>O</td>
<td>uint32_t * const data</td>
<td>Data pointer to area in which to store an accumulation data</td>
</tr>
<tr>
<td>O</td>
<td>uint16_t * const counter</td>
<td>Counter pointer to area in which to store accumulation counter 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_ARGERROR</td>
<td>The channel is out of the accumulation channel range (0 to 15)</td>
</tr>
</tbody>
</table>
R_<Config_ADCXn>_Create_UserInit

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

Remark  This API function is called as the R_<Config_ADCXn>_Create callback routine.

[Syntax]

void R_<Config_ADCXn>_Create_UserInit(void);

Remark  ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit number.

[Argument(s)]

None.

[Return value]

None.
**r_<Config_ADCXn>_error_interrupt**

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

**[Syntax]**

```c
void r_<Config_ADCXn>_error_interrupt(void);
```

Remark  
ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_ADCXn>_scan_groupx_end_interrupt**

Performs processing in response to the scan group end interrupt.

**[Syntax]**

```c
void r_<Config_ADCXn>_scan_groupx_end_interrupt(void);
```

**Remark**
ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), \( n \) is the unit number, \( x \) is the scan group number.

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

**[Return value]**
None.
**r_<Config_ADCXn>_monitor_error_interrupt**

Performs processing in response to the monitor error interrupt.

**[Syntax]**

```c
void r_<Config_ADCXn>_monitor_error_interrupt(void);
```

**Remark**  
ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_ADCXn>_sg_diag_end_interrupt**

Performs processing in response to the SG-Diag end interrupt.

**[Syntax]**

```c
void r_<Config_ADCXn>_sg_diag_end_interrupt(void);
```

**Remark**
ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit number.

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

**[Return value]**
None.
**r_<Config_ADCXn>_mpx_request_interrupt**

Performs processing in response to the MPX DMA trigger request interrupt.

**[Syntax]**

```c
void r_<Config_ADCXn>_mpx_request_interrupt(void);
```

**Remark**

ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_<Config_ADCXn>_asf_channelk_end_interrupt

Performs processing in response to the ASF accumulation end interrupt on accumulation channel k.

[Syntax]

```c
void r_<Config_ADCXn>_asf_channelk_end_interrupt(void);
```

Remark

ADCX is ADC name (ADCA, ADCC, ADCJ, ADCK), n is the unit number, k is the accumulation channel 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 3.4 API Functions: [CSI Slave]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_CSIXn&gt;_Create</td>
<td>Performs initialization necessary to control the CSI slave functions</td>
</tr>
<tr>
<td>R_&lt;Config_CSIXn&gt;_Start</td>
<td>Enables CSI</td>
</tr>
<tr>
<td>R_&lt;Config_CSIXn&gt;_Stop</td>
<td>Disables CSI</td>
</tr>
<tr>
<td>R_&lt;Config_CSIXn&gt;_Send</td>
<td>Starts data transmission</td>
</tr>
<tr>
<td>R_&lt;Config_CSIXn&gt;_Receive</td>
<td>Starts data reception</td>
</tr>
<tr>
<td>R_&lt;Config_CSIXn&gt;_Send_Receive</td>
<td>Starts data transmission and reception</td>
</tr>
<tr>
<td>R_&lt;Config_CSIXn&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the CSI slave functions</td>
</tr>
<tr>
<td>r_&lt;Config_CSIXn&gt;_interrupt_send</td>
<td>Processing is performed according to the generation of the send status interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_CSIXn&gt;_interrupt_receive</td>
<td>Processing is performed according to the generation of the receive status interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_CSIXn&gt;_interrupt_error</td>
<td>Processing is performed according to the occurrence of a communication error interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_CSIXn&gt;_callback_sendend</td>
<td>Processing is performed according to the generation of the send status interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_CSIXn&gt;_callback_receiveend</td>
<td>Processing is performed according to the generation of the receive status interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_CSIXn&gt;_callback_error</td>
<td>Processing is performed according to the occurrence of a communication error interrupt</td>
</tr>
</tbody>
</table>
R_<Config_CSI\text{Xn}>_Create

Performs initialization necessary to control the CSI slave functions.

Remark This API function is called by \texttt{R\_Systeminit} function.

\textbf{Syntax]

\begin{verbatim}
void R_<Config_CSI\text{Xn}>_Create(void);
\end{verbatim}

Remark CSI\text{X} is CSI name (CSIG, CSIH), \textit{n} is the channel number.

\textbf{Argument(s)}

None.

\textbf{Return value}

None.
R_<Config_CSI>Xn>_Start

Enables CSI.

[Syntax]
void R_<Config_CSI>Xn>_Start(void);

Remark CSIX is CSI name (CSIG, CSIH), n is the channel number.

[Argument(s)]
None.

[Return value]
None.
R_<Config_CSI\text{n}>_Stop

Disables CSI.

[Syntax]

```c
void R_\<Config_CSI\text{n}\>_Stop(void);
```

Remark CSIX is CSI name (CSIG, CSIH), \(n\) is the channel number.

[Argument(s)]

None.

[Return value]

None.
**R_<ConfigCSI><n>_Send**

Starts data transmission.

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

**Remark2.** `R_<ConfigCSI><n>_Start` must be called before this API function is called.

**[Syntax]**

```
MD_STATUS R_<ConfigCSI><n>_Send(uint16_t* const tx_buf, uint16_t tx_num);
```

**Remark** 
CSI is CSI name (CSIG, CSIH), `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* 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>
</tbody>
</table>
R_<Config_CSIXn>_Receive

Starts data reception.

Remark1. This API function performs 2 byte-level CSIX 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_CSIXn>_Start is then called.

**[Syntax]**

```c
MD_STATUS R_<Config_CSIXn>_Receive(uint16_t* const rx_buf, uint16_t rx_num);
```

Remark CSIX is CSI name (CSIG, CSIH), \( n \) 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* 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_CSIXn>_Send_Receive**

Starts data transmission and reception.

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

**Remark2.** This API function performs 2 byte-level CSIH reception the number of times specified by the argument `rx_num`, and stores the data in the buffer specified by the argument `rx_buf`.

**Remark3** `R_<Config_CSIXn>_Start` must be called before this API function is called.

**[Syntax]**

```c
MD_STATUS R_<Config_CSIXn>_Send_Receive(uint16_t* const tx_buf, uint16_t rx_num, uint16_t* const rx_buf);
```

Remark CSIX is CSI name (CSIG, CSIH), `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* const tx_buf</td>
<td>Pointer to a buffer storing the transmission data which count is same as received data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t rx_num</td>
<td>Total amount of data to receive</td>
</tr>
<tr>
<td>O</td>
<td>uint16_t* const rx_buf</td>
<td>Pointer to a buffer to store the received data</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid specification of <code>rx_num</code></td>
</tr>
</tbody>
</table>
**R_<Config_CSIXn>_Create_UserInit**

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

**Remark** This API function is called as the `R_<Config_CSIXn>_Create` callback routine.

**[Syntax]**

```c
void R_<Config_CSIXn>_Create_UserInit(void);
```

**Remark** CSI is CSI name (CSIG, CSIH), `n` is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_CSIXn>_interrupt_send**

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

**[Syntax]**

```c
void r_<Config_CSIXn>_interrupt_send(void);
```

Remark: CSI is CSI name (CSIG, CSIH), n is the channel number.

**[Argument(s)]**

None.

**[Return value]**

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

**Syntax**

```c
void r_<Config_CSIXn>_interrupt_receive(void);
```

**Remark**

CSI is CSI name (CSIG, CSIH), \(n\) 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_CSIXn>_interrupt_error(void);
```

Remark: CSI is CSI name (CSIG, CSIH), n is the channel number.

[Argument(s)]

None.

[Return value]

None.
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 \textit{r\_Config\_CSI\textit{Xn}\_interrupt\_send} corresponding to the CSIG communication interrupt.

\textbf{Syntax}

\begin{verbatim}
void r\_Config\_CSI\textit{Xn}\_callback\_sendend(void);
\end{verbatim}

Remark: CSI\textit{X} is CSI name (CSIG, CSI\textit{H}), \textit{n} is the channel number.

\textbf{Argument(s)}

None.

\textbf{Return value}

None.
**r_<Config_CSI\textit{n}>_callback_receiveend**

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  
\( r_{<\text{Config_CSI}\textit{n}>\_interrupt\_receive} \) corresponding to the CSIG reception interrupt.

**[Syntax]**

```c
void r_<Config_CSI\textit{n}>_callback_receiveend(void);
```

**Remark**  
CSI\textit{X} is CSI name (CSIG, CSI\textit{H}), \textit{n} is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
### r_<Config_CSIXn>_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 corresponding to r_<Config_CSIXn>_interrupt_error the CSIG error interrupt.

#### [Syntax]

```c
void r_<Config_CSIXn>_callback_error(uint32_t err_type);
```

**Remark**  CSI X is CSI name (CSIG, CSIH), 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 err_type</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.
Usage example

Refer to CSI master Usage example.
### 3.2.4 CSI Master

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

**Table 3.5 API Functions: [CSI Master]**

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_CSI&gt;&lt;Create&gt;</td>
<td>Performs initialization necessary to control the CSI master functions</td>
</tr>
<tr>
<td>R_&lt;Config_CSI&gt;&lt;Start&gt;</td>
<td>Enables CSI</td>
</tr>
<tr>
<td>R_&lt;Config_CSI&gt;&lt;Stop&gt;</td>
<td>Prohibit CSI</td>
</tr>
<tr>
<td>R_&lt;Config_CSI&gt;&lt;Send&gt;</td>
<td>Starts data transmission</td>
</tr>
<tr>
<td>R_&lt;Config_CSI&gt;&lt;Receive&gt;</td>
<td>Starts data reception</td>
</tr>
<tr>
<td>R_&lt;Config_CSI&gt;&lt;Send_Receive&gt;</td>
<td>Performs user-defined initialization relating to the CSI master functions</td>
</tr>
<tr>
<td>r_&lt;Config_CSI&gt;&lt;UserInit&gt;</td>
<td>Processing is performed according to the generation of the send status interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_CSI&gt;&lt;interrupt_send&gt;</td>
<td>Processing is performed according to the generation of the receive status interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_CSI&gt;&lt;interrupt_receive&gt;</td>
<td>Processing is performed according to the occurrence of a communication error interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_CSI&gt;&lt;callback_sendend&gt;</td>
<td>Processing is performed according to the generation of the send status interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_CSI&gt;&lt;callback_receiveend&gt;</td>
<td>Processing is performed according to the generation of the receive status interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_CSI&gt;&lt;callback_error&gt;</td>
<td>Processing is performed according to the occurrence of a communication error interrupt</td>
</tr>
</tbody>
</table>
R_<Config_CSIXn>_Create

Performs initialization necessary to control the CSI master functions.

Remark This API function is called by R_Systeminit function.

[Syntax]

void R_<Config_CSIXn>_Create(void);

Remark CSI X is CSI name (CSIG, CSIH), n is the channel number.

[Argument(s)]

None.

[Return value]

None.
**R_<Config_CSI\textit{n}>_Start**

Enables CSI.

**[Syntax]**

```c
void R_<Config_CSI\textit{n}>_Start(void);
```

Remark: CSIX is CSI name (CSIG, CSIH), \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_CSIXn>_Stop**

Disables CSI.

**[Syntax]**

```c
void R_<Config_CSIXn>_Stop(void);
```

**Remark**  
CSIX is CSI name (CSIG, CSIH), \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_CSIXn>_Send

Starts data transmission.

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

Remark2: R_<Config_CSIXn>_Start must be called before this API function is called.

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

[Syntax]

CSIG case

```
MD_STATUS R_<Config_CSIGn>_Send(uint16_t* const tx_buf, uint16_t tx_num);
```

Remark: n is the channel number.

CSIH case

```
MD_STATUS R_<Config_CSIHn>_Send(uint16_t* const 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* const tx_buf</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t tx_num</td>
<td>Total amount of data to send</td>
</tr>
<tr>
<td>I</td>
<td>uint32_t chip_id</td>
<td>Slave channel ID (Multiple choices are possible by using &quot;I&quot; operator)</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</td>
</tr>
</tbody>
</table>
**R_<Config_CSI>Xn>_Receive**

Starts data reception.

**Remark1**  
This API function performs 2 byte-level CSIX reception the number of times specified by the argument \textit{rx\_num}, and stores the data in the buffer specified by the argument \textit{rx\_buf}.

**Remark2**  
Starts after this API function is called, and \textit{R_<Config_CSI>Xn>_Start} is then called.

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

**[Syntax]**

**CSIG case**

```
MD_STATUS  R_<Config_CSIGN>_Receive(uint16_t* const rx_buf, uint16_t rx_num);
```

Remark  
\textit{n} is the channel number.

**CSIH case**

```
MD_STATUS  R_<Config_CSIHz>_Receive(uint16_t* const rx_buf, uint16_t rx_num, uint32_t chip_id);
```

Remark  
\textit{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* const \textit{rx_buf}</td>
<td>Pointer to a buffer to store the received data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t \textit{rx_num}</td>
<td>Total amount of data to receive</td>
</tr>
<tr>
<td></td>
<td>uint32_t \textit{chip_id}</td>
<td>Slave channel ID (Multiple choices are possible by using &quot;I&quot; operator) _CSIH_SELECT_CHIP_n: Channel \textit{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 \textit{rx_num}</td>
</tr>
</tbody>
</table>
R_<Config_CSIXn>_Send_Receive

Starts data transmission and reception.

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

Remark2 This API function performs 2 byte-level CSIH reception the number of times specified by the argument tx_num, and stores the data in the buffer specified by the argument rx_buf.

Remark3 R_<Config_CSIXn>_Start must be called before this API function is called.

[Syntax]

CSIG case

MD_STATUS R_<Config_CSIGn>_Send_Receive(uint16_t* const tx_buf, uint16_t tx_num, uint16_t* const rx_buf);

Remark n is the channel number.

CSIH case

MD_STATUS R_<Config_CSIHn>_Send_Receive(uint16_t* const tx_buf, uint16_t tx_num, uint16_t* const rx_buf, 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* const tx_buf</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t tx_num</td>
<td>Total amount of data to send</td>
</tr>
<tr>
<td>O</td>
<td>uint16_t* const rx_buf</td>
<td>Pointer to a buffer to store the received data which count is same as sent data.</td>
</tr>
<tr>
<td>I</td>
<td>uint32_t chip_id</td>
<td>Slave channel: _CSIH_SELECT_CHIP_n; 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</td>
</tr>
</tbody>
</table>
**R_<ConfigCSIXn>_Create_UserInit**

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

**Remark**  This API function is called as the R_<ConfigCSIXn>_Create callback routine.

**[Syntax]**

```c
void R_<ConfigCSIXn>_Create_UserInit(void);
```

**Remark**  CSI[X] is CSI name (CSIG, CSIH), n is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_CSI\text{n}>_interrupt_send**

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

**[Syntax]**

```c
void _r_<Config_CSI\text{n}>_interrupt_send(void);
```

Remark: CSI\text{X} is CSI name (CSIG, CSIH), \text{n} is the channel number.

**[Argument(s)]**

None.

**[Return value]**

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

**Syntax**

```c
void r_<ConfigCSI><n>_interrupt_receive(void);
```

Remark: CSIX is CSI name (CSIG, CSIH), n is the channel number.

**Argument(s)**

None.

**Return value**

None.
**r_given\langle ConfigCSI\rangle_xn\rangle\_interrupt\_error**

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

**[Syntax]**

```c
void r_given\langle ConfigCSI\rangle_xn\rangle\_interrupt\_error(void);
```

**Remark**

CSI is CSI name (CSIG, CSIH), \(n\) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
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_CSIXn>_interrupt_send` corresponding to the CSIG communication interrupt.

### Syntax

```c
void r_<Config_CSIXn>_callback_sendend(void);
```

**Remark** CSIX is CSI name (CSIG, CSIH), `n` is the channel number.

### Argument(s)

None.

### Return value

None.
**r_<Config_CSIXn>_callback_receiveend**

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_CSIXn>_interrupt_receive corresponding to the CSIG reception interrupt.

**[Syntax]**

```c
void r_<Config_CSIXn>_callback_receiveend(void);
```

**Remark**  CSI X is CSI name (CSIG, CSIH), n is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_CSIXn>_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_CSIXn>_interrupt_error` corresponding to the CSIG error interrupt.

**[Syntax]**

```
void r_<Config_CSIXn>_callback_error(uint32_t err_type);
```

**Remark**  
CSI is CSI name (CSIG, CSIH), `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 CSIG error interrupt.  
     |              | 0000x0x1B: Overrun error  
     |              | 0000x01xB: Parity error  
     |              | 000010xxB: Data consistency check error |

**[Return value]**

None.
Usage example

This is an example for CSIH1 as master to send/receive data with CSIH0 as slave:
(Blue code is user code.)

```
/* Start user code for global. Do not edit comment generated here */
uint16_t tx_buf_h1[] = {0xA5A5, 0x5A5A};
uint16_t rx_buf1_h1[] = {0x8888, 0x8888};
uint16_t rx_buf2_h1[] = {0x9999, 0x9999};

uint16_t tx_buf_h0[] = {0xCACA, 0xBABA};
uint16_t rx_buf1_h0[] = {0x8888, 0x8888};
uint16_t rx_buf2_h0[] = {0x9999, 0x9999};
/* End user code. Do not edit comment generated here */

void main(void)
{
    r_main_userinit();
    /* Start user code for main. Do not edit comment generated here */
    R_Config_CSIH0_Start();
    R_Config_CSIH1_Start();
    R_Config_CSIH0_Send_Receive(tx_buf_h0, 2, rx_buf1_h0);
    R_Config_CSIH1_Send_Receive(tx_buf_h1, 2, rx_buf1_h1, _CSIH_SELECT_CHIP_1);
    while(transmitend_flag1 != 1);
    while(receiveend_flag1 != 1);
    transmitend_flag1 = 0;
    receiveend_flag1 = 0;
    transmitend_flag0 = 0;
    receiveend_flag0 = 0;
    R_Config_CSIH0_Send_Receive(tx_buf_h0, 2, rx_buf2_h0);
    R_Config_CSIH1_Send_Receive(tx_buf_h1, 2, rx_buf2_h1, _CSIH_SELECT_CHIP_1);
    while(transmitend_flag1 != 1);
    while(receiveend_flag1 != 1);
    transmitend_flag1 = 0;
    receiveend_flag1 = 0;
    transmitend_flag0 = 0;
    receiveend_flag0 = 0;
    /* End user code. Do not edit comment generated here */
}
```

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t transmitend_flag_0 = 0;
volatile uint8_t receiveend_flag_0 = 0;
/* End user code. Do not edit comment generated here */

void r_Config_CSIH1_callback_sendend(void)
{
    /* Start user code for r_Config_CSIH1_callback_sendend. Do not edit comment generated here */
    transmitend_flag1 = 1;
    /* End user code. Do not edit comment generated here */
}
```
void r_Config_CSIH1_callback_receiveend(void)  
{
    /* Start user code for r_Config_CSIH1_callback_receiveend. Do not edit comment generated here */
    receiveend_flag1 = 1;
    /* End user code. Do not edit comment generated here */
}

Config_CSIH0_user.c

/* Start user code for global. Do not edit comment generated here */
volatile uint8_t transmitend_flag0 = 0;
volatile uint8_t receiveend_flag0 = 0;
/* End user code. Do not edit comment generated here */

void r_Config_CSIH0_callback_sendend(void)  
{
    /* Start user code for r_Config_CSIH0_callback_sendend. Do not edit comment generated here */
    transmitend_flag0 = 1;
    /* End user code. Do not edit comment generated here */
}

void r_Config_CSIH0_callback_receiveend(void)  
{
    /* Start user code for r_Config_CSIH0_callback_receiveend. Do not edit comment generated here */
    receiveend_flag0 = 1;
    /* End user code. Do not edit comment generated here */
}
### 3.2.5 Interrupt

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R._&lt;Config_INTC&gt;_Create</td>
<td>Performs initialization necessary to control the interrupt functions</td>
</tr>
<tr>
<td>R._&lt;Config_INTC&gt;_INTPn_Start</td>
<td>Enables the INTPn interrupts</td>
</tr>
<tr>
<td>R._&lt;Config_INTC&gt;_INTPn_Stop</td>
<td>Disables the INTPn interrupts</td>
</tr>
<tr>
<td>R._&lt;Config_INTC&gt;_IRQn_Start</td>
<td>Enables the IRQn interrupts</td>
</tr>
<tr>
<td>R._&lt;Config_INTC&gt;_IRQn_Stop</td>
<td>Disables the IRQn interrupts</td>
</tr>
<tr>
<td>R._&lt;Config_INTC&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the interrupt functions</td>
</tr>
<tr>
<td>r._&lt;Config_INTC&gt;_intpn_interrupt</td>
<td>Performs processing in response to the INTPn interrupt</td>
</tr>
<tr>
<td>r._&lt;Config_INTC&gt;_intpn_interrupt_pek</td>
<td>Performs processing in response to the INTPn interrupt for PEk</td>
</tr>
<tr>
<td>r._&lt;Config_INTC&gt;_irqn_interrupt</td>
<td>Performs processing in response to the IRQn interrupt</td>
</tr>
<tr>
<td>r._&lt;Config_INTC&gt;_nmi_interrupt</td>
<td>Performs processing in response to the NMI interrupt</td>
</tr>
<tr>
<td>R._&lt;Config_INTC&gt;_NMI_Start</td>
<td>Enables NMI interrupt</td>
</tr>
<tr>
<td>R._&lt;Config_INTC&gt;_NMI_Stop</td>
<td>Disables NMI interrupt</td>
</tr>
<tr>
<td>R._&lt;Config_INTC&gt;_SINTn_Start</td>
<td>Enables the SINTn interrupt</td>
</tr>
<tr>
<td>R._&lt;Config_INTC&gt;_SINTn_Stop</td>
<td>Disables the SINTn interrupt</td>
</tr>
<tr>
<td>R._&lt;Config_INTC&gt;_SINTn_TriggerOn</td>
<td>Triggers SINTn interrupt</td>
</tr>
<tr>
<td>r._&lt;Config_INTC&gt;_sintn_interrupt_pek</td>
<td>Performs processing in response to the SINTn interrupt for PEk</td>
</tr>
</tbody>
</table>
R_<Config_INTC>_Create

Performs initialization necessary to control the interrupt functions.

Remark This API function is called by R_Systeminit function.

[Syntax]

```
void R_<Config_INTC>_Create(void);
```

[Argument(s)]

None.

[Return value]

None.
R_<Config_INTC>_INTPn_Start

Enables the INTPn interrupts.

[Syntax]

```c
void R_<Config_INTC>_INTPn_Start(void);
```

Remark  

$n$ is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.
R_<Config_INTC>_INTPn_Stop

Disables the INTPn interrupts.

[Syntax]

void R_<Config_INTC>_INTPn_Stop(void);

Remark $n$ is the interrupt factor number.

[Argument(s)]
None.

[Return value]
None.
R_<Config_INTC>_IRQn_Start

Enables the IRQn interrupts.

[Syntax]

```c
void R_<Config_INTC>_IRQn_Start(void);
```

Remark  

n is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.
**R_<Config_INTC>_IRQn_Stop**

Disables the IRQn interrupts.

**[Syntax]**

```c
void R_<Config_INTC>_IRQn_Stop(void);
```

Remark  

$n$ is the interrupt factor number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_INTC>_Create_UserInit**

Performs user-defined initialization relating to the interrupt functions.

**Remark**  
This API function is called as the `R_<Config_INTC>_Create` callback routine.

**[Syntax]**

```c
void  R_<Config_INTC>_Create_UserInit(void);
```

**[Argument(s)]**

None.

**[Return value]**

None.
r_<Config_INTC>_intp_n_interrupt

Performs processing in response to the INTP\textsubscript{n} interrupt.

**[Syntax]**

```c
void r_<Config_INTC>_intp_n_interrupt(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_<Config_INTC>_intp_n_interrupt_pek

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

**[Syntax]**

```c
void r_<Config_INTC>_intp_n_interrupt_pek(void);
```

Remark \(n\) is the interrupt factor number, \(k\) is the CPU core number.

**[Argument(s)]**

None.

**[Return value]**

None.
Performs processing in response to the Irqn interrupt.

### Syntax

```c
void r_<Config_INTC>_irqn_interrupt(void);
```

Remark  
- `n` is the interrupt factor number.

### Argument(s)

None.

### Return value

None.
**r_<Config_INTC>_nmi_interrupt**

Performs processing in response to the NMI interrupt.

**[Syntax]**

```c
void r_<Config_INTC>_nmi_interrupt(void);
```

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_INTC>_NMI_Start**

Enables the NMI interrupts.

**[Syntax]**

```c
void R_<Config_INTC>_NMI_Start(void);
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_\(<\text{Config\_INTC}>\)_NMI\_Stop

Disables the NMI interrupts.

[Syntax]

```
void R_\(<\text{Config\_INTC}>\)_NMI\_Stop(void);
```

[Argument(s)]

None.

[Return value]

None.
**R\_<Config\_INTC>_SINTn\_Start**

Enables the SINT\(n\) interrupts.

**[Syntax]**

```c
void R\_<Config\_INTC>_SINTn\_Start(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_INTC>_SINTn_Stop**

Disables the SINT\(n\) interrupts.

**[Syntax]**

```c
void R_<Config_INTC>_SINTn_Stop(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_INTC>_SINTn_TriggerOn

Triggers the SINTn interrupts.

[Syntax]

void R_<Config_INTC>_SINTn_TriggerOn(void);

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

[Argument(s)]
None.

[Return value]
None.
r_<Config_INTC>_sintn_interrupt_pek

Performs processing in response to the SINTn interrupt.

[Syntax]

void r_<Config_INTC>_sintn_interrupt_pek(void);

Remark \( n \) is the software interrupt number, \( k \) is the CPU core number.

[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 3.7  API Functions: [Input Interval Timer]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_\textless Config_TAUXn_\textgreater_Create</td>
<td>Performs initialization necessary to control the input interval timer functions</td>
</tr>
<tr>
<td>R_\textless Config_TAUXn_\textgreater_Start</td>
<td>Starts the count for channel $m$</td>
</tr>
<tr>
<td>R_\textless Config_TAUXn_\textgreater_Stop</td>
<td>Ends the count for channel $m$</td>
</tr>
<tr>
<td>R_\textless Config_TAUXn_\textgreater_Create_UserInit</td>
<td>Performs user-defined initialization relating to the input interval timer functions</td>
</tr>
<tr>
<td>r_\textless Config_TAUXn_\textgreater_interrupt</td>
<td>Performs processing in response to the timer interrupt</td>
</tr>
<tr>
<td>r_\textless Config_TAUXn_\textgreater_interrupt_peK</td>
<td>Performs processing in response to the timer interrupt for PE$k$</td>
</tr>
</tbody>
</table>
**R_<Config_TAUx_n_m>_Create**

Performs initialization necessary to control the input interval timer functions.

Remark This API function is called by **R_Systeminit** function.

**[Syntax]**

```c
void R_<Config_TAUx_n_m>_Create(void);
```

Remark TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_TAUXn_m>_Start**

Starts the count for channel \( m \).

**[Syntax]**

```c
void R_<Config_TAUXn_m>_Start(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_TAUXn_m>_Stop**

Ends the count for channel \( m \).

**[Syntax]**

```c
void R_<Config_TAUXn_m>_Stop(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_TAUXn_m>_Create_UserInit**

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

**Remark** This API functions is called as the **R_<Config_TAUXn_m>_Create** callback routine.

**[Syntax]**

```c
void R_<Config_TAUXn_m>_Create_UserInit(void);
```

**Remark** TAU is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
### r\_<Config_TAU\_n\_m>\_interrupt

Performs processing in response to the timer interrupt.

#### Syntax

```c
void r\_<Config_TAU\_n\_m>\_interrupt(void);
```

**Remark**

TAU is TAU name (TAUB, TAUD, TAUJ), \(n\) is the unit number, \(m\) is the channel number.

#### Argument(s)

None.

#### Return value

None.
**r_<Config_TAUXn_m>_interrupt_pek**

Performs processing in response to the timer interrupt for PEk.

**[Syntax]**

```c
void r_<Config_TAUXn_m>_interrupt_pek(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number, \( k \) is the PE 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 3.8 API Functions: [Input Pulse Interval Measurement]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_TAUxn_m&gt;_Create</td>
<td>Performs initialization necessary to control the input pulse interval measurement functions</td>
</tr>
<tr>
<td>R_&lt;Config_TAUxn_m&gt;_Start</td>
<td>Starts the count for channel m</td>
</tr>
<tr>
<td>R_&lt;Config_TAUxn_m&gt;_Stop</td>
<td>Ends the count for channel m</td>
</tr>
<tr>
<td>R_&lt;Config_TAUxn_m&gt;_Get_PulseWidth</td>
<td>Reads the input pulse width of the timer</td>
</tr>
<tr>
<td>R_&lt;Config_TAUxn_m&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the input pulse interval measurement functions</td>
</tr>
<tr>
<td>r_&lt;Config_TAUxn_m&gt;_interrupt</td>
<td>Performs processing in response to the timer interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_TAUxn_m&gt;_interrupt_pek</td>
<td>Performs processing in response to the timer interrupt for PEk</td>
</tr>
</tbody>
</table>
**R_<Config_TAU>Xn_m>_Create**

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

**Remark**  This API function is called by **R_Systeminit** function.

**[Syntax]**

```c
void R_<Config_TAU>Xn_m>_Create(void);
```

**Remark**  TAU is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TAUXn_m>_Start

Starts the count for channel \( m \).

**[Syntax]**

```c
void R_<Config_TAUXn_m>_Start(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TAUXn_m>_Stop

Ends the count for channel \( m \).

**[Syntax]**

```c
void R_<Config_TAUXn_m>_Stop(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_TAU\_Xn\_m>\_Get\_PulseWidth**

Reads the input pulse width of the timer.

**[Syntax]**

```c
void R_<Config_TAU\_Xn\_m>\_Get\_PulseWidth(uint32_t * const width);
```

Remark: TAU is TAU name (TAUB, TAUD, TAUJ), 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_TAUXn_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_TAUXn_m>_Create callback routine.

**[Syntax]**

```c
void R_<Config_TAUXn_m>_Create_UserInit(void);
```

**Remark**  TAU is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_TAUXn_m>_interrupt**

Performs processing in response to the timer interrupt.

**[Syntax]**

```c
void r_<Config_TAUXn_m>_interrupt(void);
```

Remark: TAU is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_TAUXn_m>_interrupt_pek**

Performs processing in response to the timer interrupt for PEk.

**[Syntax]**

```c
void r_<Config_TAUXn_m>_interrupt_pek(void);
```

Remark: TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number, k is the PE 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 3.9 API Functions: [Interval Timer]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{R_{&lt;Config_TAUXn_m&gt;_Create}}</td>
<td>Performs initialization necessary to control the interval timer functions</td>
</tr>
<tr>
<td>\texttt{R_{&lt;Config_TAUXn_m&gt;_Start}}</td>
<td>Starts the count for channel ( m )</td>
</tr>
<tr>
<td>\texttt{R_{&lt;Config_TAUXn_m&gt;_Stop}}</td>
<td>Ends the count for channel ( m )</td>
</tr>
<tr>
<td>\texttt{R_{&lt;Config_TAUXn_m&gt;_Create_UserInit}}</td>
<td>Performs user-defined initialization relating to the interval timer functions</td>
</tr>
<tr>
<td>\texttt{r_{&lt;Config_TAUXn_m&gt;_interrupt}}</td>
<td>Performs processing in response to the timer interrupt</td>
</tr>
<tr>
<td>\texttt{r_{&lt;Config_TAUXn_m&gt;_interrupt_pek}}</td>
<td>Performs processing in response to the timer interrupt for PEk</td>
</tr>
</tbody>
</table>
R_<Config_TAUxn_m>_Create

Performs initialization necessary to control the interval timer functions.

Remark    This API function is called by R_Systeminit function.

[Syntax]
void R_<Config_TAUxn_m>_Create(void);

Remark    TAU is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

[Argument(s)]
None.

[Return value]
None.
**R_\langle Config\_TAUXn\_m\rangle\_Start**

Starts the count for channel \( m \).

**[Syntax]**

```c
void R_\langle Config\_TAUXn\_m\rangle\_Start(void);
```

**Remark**  
TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R\_\_Config\_TAUXn\_m\_Stop**

Ends the count for channel \( m \).

**[Syntax]**

```c
void R_{Config\_TAUXn\_m}_Stop(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_**<Config_TAUXn_m>_Create_UserInit

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

**Remark**

This API function is called as the R_**<Config_TAUXn_m>_Create** callback routine.

**[Syntax]**

```c
void R_**<Config_TAUXn_m>_Create_UserInit**(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_\langle Config\_TAUXn\_m\rangle\_interrupt**

Performs processing in response to the timer interrupt.

**[Syntax]**

```c
void r_\langle Config\_TAUXn\_m\rangle\_interrupt(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_TAUXn_m>_interrupt_pek**

Performs processing in response to the timer interrupt for PE_k.

**[Syntax]**

```c
void r_<Config_TAUXn_m>_interrupt_pek(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number, k is the PE 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 3.10 API Functions: [Triangle PWM Output]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_TAUx&gt;_Create</td>
<td>Performs initialization necessary to control the triangle PWM output functions</td>
</tr>
<tr>
<td>R_&lt;Config_TAUx&gt;_Start</td>
<td>Starts timer counting</td>
</tr>
<tr>
<td>R_&lt;Config_TAUx&gt;_Stop</td>
<td>Stops timer counting</td>
</tr>
<tr>
<td>R_&lt;Config_TAUx&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the triangle PWM output functions</td>
</tr>
<tr>
<td>r_&lt;Config_TAUx&gt;_channelm_interrupt</td>
<td>Performs processing in response to the timer interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_TAUx&gt;_channelm_interrupt_pek</td>
<td>Performs processing in response to the timer interrupt for PEk</td>
</tr>
</tbody>
</table>
R_<Config_TAUXn>_Create

Performs initialization necessary to control the triangle PWM output functions.

Remark  This API function is called by R_Systeminit function.

**[Syntax]**

```c
void R_<Config_TAUXn>_Create(void);
```

Remark  TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TAUxn>_Start

Starts timer counting.

**[Syntax]**

```c
void R_<Config_TAUxn>_Start(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
Stops timer counting.

**[Syntax]**

```c
void R_<Config_TAUxn>_Stop(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_TAUxn>_Create_UserInit**

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

**Remark** This API function is called as the `R_<Config_TAUxn>_Create` callback routine.

**[Syntax]**

```c
void R_<Config_TAUxn>_Create_UserInit(void);
```

**Remark** TAU is TAU name (TAUB, TAUD, TAUJ), `n` is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_TAUXn>_channelm_interrupt**

Performs processing in response to the timer interrupt.

**[Syntax]**

```c
void r_<Config_TAUXn>_channelm_interrupt(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_TAUxn>_channelm_interrupt_pek**

Performs processing in response to the timer interrupt for PE\textsubscript{k}.

**[Syntax]**

```c
void r_<Config_TAUxn>_channelm_interrupt_pek(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), \textit{n} is the unit number, \textit{m} is the channel number, \textit{k} is the PE number.

**[Argument(s)]**

None.

**[Return value]**

None.
### 3.2.10 Triangle PWM Output With Dead Time

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

**Table 3.10** API Functions: [Triangle PWM Output With Dead Time]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_TAUx&gt;_Create</td>
<td>Performs initialization necessary to control the triangle PWM output with dead time functions</td>
</tr>
<tr>
<td>R_&lt;Config_TAUx&gt;_Start</td>
<td>Starts the count for channel m</td>
</tr>
<tr>
<td>R_&lt;Config_TAUx&gt;_Stop</td>
<td>Ends the count for channel m</td>
</tr>
<tr>
<td>R_&lt;Config_TAUx&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the triangle PWM output with dead time functions</td>
</tr>
<tr>
<td>r_&lt;Config_TAUx&gt;_channelm_interrupt</td>
<td>Performs processing in response to the timer interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_TAUx&gt;_channelm_interrupt_pek</td>
<td>Performs processing in response to the timer interrupt for PEk</td>
</tr>
</tbody>
</table>
**R_<Config_TAUXn>_Create**

Performs initialization necessary to control the triangle PWM output with dead time functions.

**Remark**  This API function is called by **R_Systeminit** function.

**[Syntax]**

```c
void _R_Config_TAUXn_Create(void);
```

**Remark**  TAUX is TAU name (TAUB, TAUD, TAUJ), *n* is the unit number.

**[Argument(s)]

None.

**[Return value]

None.
R_<Config_TAUXn>_Start

Starts the counting.

**[Syntax]**

```c
void R_<Config_TAUXn>_Start(void);
```

**Remark** TAU is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TAUXn>_Stop

Ends the counting.

[Syntax]
void R_<Config_TAUXn>_Stop(void);

Remark TAU is TAU name (TAUB, TAUD, TAUJ), n is the unit number.

[Argument(s)]
None.

[Return value]
None.
**R_<Config_TAUXn>_Create_UserInit**

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

**Remark**  This API functions is called as the **R_<Config_TAUXn>_Create** callback routine.

**[Syntax]**

```c
void R_<Config_TAUXn>_Create_UserInit(void);
```

**Remark**  TAU is TAU name (TAUB, TAUD, TAUJ), n is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_TAUXn>_channelm_interrupt**

Performs processing in response to the timer interrupt.

**[Syntax]**

```c
void r_<Config_TAUXn>_channelm_interrupt(void);
```

**Remark**
TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

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

**[Return value]**
None.
**r_<Config_TAUXn>__channelm_interrupt_pek**

Performs processing in response to the timer interrupt for PEk

**[Syntax]**

```c
void r_<Config_TAUXn>__channelm_interrupt_pek(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number, k is the PE number.

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.11 OS Timer

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

Table 3.11 API Functions: [OS Timer]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R.&lt;Config_OSTMn&gt;_Create</td>
<td>Performs initialization necessary to control the OS timer functions</td>
</tr>
<tr>
<td>R.&lt;Config_OSTMn&gt;_Start</td>
<td>Starts OS timer count</td>
</tr>
<tr>
<td>R.&lt;Config_OSTMn&gt;_Stop</td>
<td>Stop OS timer count</td>
</tr>
<tr>
<td>R.&lt;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.&lt;Config_OSTMn&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the OS timer functions</td>
</tr>
<tr>
<td>r.&lt;Config_OSTMn&gt;_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 by **R_Systeminit** 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**

Starts OS timer count.

**[Syntax]**

```c
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]**

```c
void R_<Config_OSTMn>_Stop(void);
```

*Remark*  

\( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_OSTM{n}_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_OSTM{n}_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]

void r_<Config_OSTMn>_interrupt(void);

Remark $n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
3.2.12 Port

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

**Table 3.12 API Functions: [Port]**

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_PORT&gt;_Create</td>
<td>Performs initialization necessary to control the I/O port functions</td>
</tr>
<tr>
<td>R_&lt;Config_PORT&gt;_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 by **R_Systeminit** function.

**[Syntax]**

```c
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 functions is called as the **R\_</Config\_PORT>_Create** callback routine.

**[Syntax]**

```c
void R\_</Config\_PORT>_Create\_UserInit(void);
```

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.13 PWM Output

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

Table 3.13 API Functions: [PWM Output]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_TAUxn&gt;_Create</td>
<td>Performs initialization necessary to control the PWM output functions</td>
</tr>
<tr>
<td>R_&lt;Config_TAUxn&gt;_Start</td>
<td>Starts timer counting</td>
</tr>
<tr>
<td>R_&lt;Config_TAUxn&gt;_Stop</td>
<td>Stops timer counting</td>
</tr>
<tr>
<td>R_&lt;Config_TAUxn&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the PWM output functions</td>
</tr>
<tr>
<td>r_&lt;Config_TAUxn&gt;_channelm_interrupt</td>
<td>Performs processing in response to the timer interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_TAUxn&gt;_channelm_interrupt_pek</td>
<td>Performs processing in response to the timer interrupt for PEk</td>
</tr>
</tbody>
</table>
**R_<Config_TAU>Xn>_Create**

Performs initialization necessary to control the PWM output functions.

**Remark**
This API function is called by **R_Systeminit** function.

**[Syntax]**

```c
void R_<Config_TAU>Xn>_Create(void);
```

**Remark**
TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_\text{Config\_TAUXn\_Start}

Starts timer counting.

**[Syntax]**

```c
void R_\text{Config\_TAUXn\_Start}(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_TAU\_Xn>_Stop**

Stops timer counting.

[Syntax]

```c
void R_<Config_TAU\_Xn>_Stop(void);
```

Remark  
TAU is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number.

[Argument(s)]

None.

[Return value]

None.
**R_{<Config\_TAUXn>\_Create\_UserInit}**

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

**Remark**  This API functions is called as the `R_{<Config\_TAUXn>\_Create}` callback routine.

**[Syntax]**

```c
void R_{<Config\_TAUXn>\_Create\_UserInit}(void);
```

**Remark**  TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
Performs processing in response to the timer interrupt.

[Syntax]

```c
void r_<Config_TAU>n_channelm_interrupt(void);
```

Remark: TAU is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

[Argument(s)]

None.

[Return value]

None.
r_<Config_TAUXn>_channelm_interrupt_pek

Performs processing in response to the timer interrupt for PEk

**[Syntax]**

```c
void r_<Config_TAUXn>_channelm_interrupt_pek(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number, k is the PE number.

**[Argument(s)]**

None.

**[Return value]**

None.
### 3.2.14 Stand-by Controller

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

**Table 3.14 API Functions: [Stand-by Controller]**

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_STBC&gt;_Prepare_Stop_Mode</td>
<td>Performs user-defined processing relating to the preparation to start stand-by (STOP mode)</td>
</tr>
<tr>
<td>R_&lt;Config_STBC&gt;_Start_Stop_Mode</td>
<td>Starts stand-by (STOP mode)</td>
</tr>
<tr>
<td>R_&lt;Config_STBC&gt;_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_&lt;Config_STBC&gt;_Start_Deep_Stop_Mode</td>
<td>Starts stand-by (DeepSTOP mode)</td>
</tr>
<tr>
<td>R_&lt;Config_STBC&gt;_Deep_Stop_Loop</td>
<td>Performs wait processing of stand-by (DeepSTOP mode)</td>
</tr>
<tr>
<td>R_&lt;Config_STBC&gt;_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_&lt;Config_STBC&gt;_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_&lt;Config_STBC&gt;_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_&lt;Config_STBC&gt;_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_&lt;Config_STBC&gt;_Prepare_Stop_Mode_Set_CP_UCLK</td>
<td>Performs user-defined processing relating to the preparation (CPU clock setting) to start stand-by (STOP mode)</td>
</tr>
<tr>
<td>R_&lt;Config_STBC&gt;_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_&lt;Config_STBC&gt;_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>
<tr>
<td>R_&lt;Config_STBC&gt;_Prepare_Deep_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 (DeepSTOP mode)</td>
</tr>
<tr>
<td>R_&lt;Config_STBC&gt;_Prepare_Deep_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 (DeepSTOP mode)</td>
</tr>
<tr>
<td>R_&lt;Config_STBC&gt;_Set_Module_XXXX_Standby_Mode</td>
<td>Set module standby mode by stopping clock connected to XXXX channel</td>
</tr>
<tr>
<td>R_&lt;Config_STBC&gt;_Cancel_Module_XXXX_Standby_Mode</td>
<td>Cancel module standby mode by supplying clock connected to XXXX channel</td>
</tr>
<tr>
<td>R_&lt;Config_STBC&gt;_Check_Module_XXXX_Idle_State</td>
<td>Check if module XXXX is in idle status before setting module standby 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

Starts stand-by (STOP mode).

Remark  R_<Config_STBC>_Prepare_Stop_Mode must be called before this API function is called.

[Syntax]

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**

Starts 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]
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>_Prepare_Stop_Mode_Set_CPUCLK**

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

**Syntax**

```c
void R_<Config_STBC>_Prepare_Stop_Mode_Set_CPUCLK(void);
```

**Argument(s)**

None.

**Return value**

None.
R_<Config_STBC>_Prepare_Deep_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_\textless Config\_STBC\textgreater\_Prepare\_Deep\_Stop\_Mode\_Set\_Interrupt

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

**[Syntax]**

\begin{verbatim}
void R_\textless Config\_STBC\textgreater\_Prepare\_Deep\_Stop\_Mode\_Set\_Interrupt(void);
\end{verbatim}

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_STBC>_Prepare_Deep_Stop_Mode_Set_Clock_Mask

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

[Syntax]

```c
void R_<Config_STBC>_Prepare_Deep_Stop_Mode_Set_Clock_Mask(void);
```

[Argument(s)]

None.

[Return value]

None.
**R_<Config_STBC>_Prepare_Deep_Stop_Mode_Set_Clock_Source**

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

**[Syntax]**

```c
void R_<Config_STBC>_Prepare_Deep_Stop_Mode_Set_Clock_Source(void);
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_STBC>_Set_Module_XXXX_Standby_Mode

Performs module standby mode setting by stopping clock connected to XXXX channel.

[Syntax]

```
MD_STATUS R_<Config_STBC>_Set_Module_XXXX_Standby_Mode(uint8_t channel);
```

Remark

XXXX represent the module/function in Table 3.14.1 XXXX module/function name

<table>
<thead>
<tr>
<th>XXXX name</th>
<th>XXXX name</th>
<th>XXXX name</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSCFD</td>
<td>MMCA</td>
<td>TAUD</td>
</tr>
<tr>
<td>FLXA</td>
<td>ENCA</td>
<td>TAUJ_ISO</td>
</tr>
<tr>
<td>GTM</td>
<td>PSI5</td>
<td>TPBA</td>
</tr>
<tr>
<td>ETNB</td>
<td>PSI5S</td>
<td>TSG3</td>
</tr>
<tr>
<td>RSENT</td>
<td>PWMD</td>
<td>OSTM</td>
</tr>
<tr>
<td>MSPi</td>
<td>RHSIF</td>
<td>ADCJ_AWO</td>
</tr>
<tr>
<td>RLIN3</td>
<td>RIIC</td>
<td>RTCA</td>
</tr>
<tr>
<td>ADCJ_ISO</td>
<td>SCI3</td>
<td>TAUJ_AWO</td>
</tr>
<tr>
<td>CXPI</td>
<td>TAPA</td>
<td></td>
</tr>
</tbody>
</table>

[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 channel</td>
<td>XXXX channel</td>
</tr>
</tbody>
</table>

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Set module standby mode successfully</td>
</tr>
<tr>
<td>MD_BUSY1</td>
<td>Reset execution of module XXXX is being processed</td>
</tr>
</tbody>
</table>
**R_<Config_STBC>_Cancel_Module_ XXXX_Standby_Mode**

Performs module standby mode canceling by supplying clock connected to XXXX channel.

**[Syntax]**

```c
MD_STATUS R_<Config_STBC>_Cancel_Module_XXXX_Standby_Mode(uint8_t channel);
```

Remark XXXX represent the module/function in 3.2.14.1 XXXX module/function name

**[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 channel</td>
<td>XXXX channel</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Set module standby mode successfully</td>
</tr>
<tr>
<td>MD_BUSY1</td>
<td>Reset execution of module XXXX is being processed</td>
</tr>
</tbody>
</table>
**R_<Config_STBC>_Check_Module_##_Idle_State**

Performs XXXX module idle state checking before setting module standby mode.

Remark  This API is called by R_<Config_STBC>_Set_Module_##_Standby_Mode.

**[Syntax]**

```c
void R_<Config_STBC>_Check_Module_##_Idle_State(void);
```

Remark  XXXX represent the module/function in Table 3.14.1 XXXX module/function name

**[Argument(s)]**

None.

**[Return value]**

None.
Usage example

This is an example for STBC transition to stop mode and wake-up by using INTP12:
(Blue code is user code.)

```c
void main(void)
{
    r_main_userinit();
    /* Start user code for main. Do not edit comment generated here */
    R_Config_INTC_INTPT12_Start();
    R_Config_STBC_Prepare_Stop_Mode();
    R_Config_STBC_Start_Stop_Mode();

    while(1);
    /* End user code. Do not edit comment generated here */
}
```
3.2.15 UART Interface

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

Table 3.15 API Functions: [UART Interface]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_UARTn&gt;_Create</td>
<td>Performs initialization necessary to control the UART interface functions</td>
</tr>
<tr>
<td>R_&lt;Config_UARTn&gt;_Start</td>
<td>Sets UART communication to standby mode</td>
</tr>
<tr>
<td>R_&lt;Config_UARTn&gt;_Stop</td>
<td>Ends UART communication</td>
</tr>
<tr>
<td>R_&lt;Config_UARTn&gt;_Send</td>
<td>Starts UART data transmission</td>
</tr>
<tr>
<td>R_&lt;Config_UARTn&gt;_Receive</td>
<td>Starts UART data reception</td>
</tr>
<tr>
<td>R_&lt;Config_UARTn&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the UART interface functions</td>
</tr>
<tr>
<td>r_&lt;Config_UARTn&gt;_interrupt_send</td>
<td>Performs processing in response to the UART communication interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_UARTn&gt;_interrupt_receive</td>
<td>Performs processing in response to the UART reception interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_UARTn&gt;_interrupt_error</td>
<td>Performs processing in response to the UART error interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_UARTn&gt;_callback_sendend</td>
<td>Performs processing in response to the UART communication interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_UARTn&gt;_callback_receiveend</td>
<td>Performs processing in response to the UART reception interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_UARTn&gt;_callback_error</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 by **R_Systeminit** 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**

Starts 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.

**Remark3**  For continuous transmissions, user should check the previous transmission is completed in advance, then call this function.

**[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

Starts 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.

Remark3 For continuous receptions, user should check the previous reception is completed in advance, then call this function.

[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 functions 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_\langle Config_UARTn\rangle\_interrupt\_error**

Performs processing in response to the UART error interrupt.

**[Syntax]**

```c
void r_\langle Config_UARTn\rangle\_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
          r_<Config_UARTn>_interrupt_send corresponding to the UART communication interrupt.

[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]

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>
<tbody>
<tr>
<td>I</td>
<td>uint32_t err_type</td>
<td>Trigger for UART error interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x00xx01B: Bit error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x00x10xB: Overrun error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x001x0xB: Framing error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0100xx0xB: Parity error</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
Usage example

This is an example for UART0 and UART1 to transmit and receive data each other with full duplex:
(Blue code is user code.)

```c
// Blue code is user code.

r_cg_main.c
extern uint8_t transmitend_flag_0;
extern uint8_t transmitend_flag_1;
extern uint8_t receiveend_flag_0;
extern uint8_t receiveend_flag_1;
uint8_t tx_buf0[3] = {0x11, 0x22, 0x33};
uint8_t tx_buf1[3] = {0x44, 0x55, 0x66};
uint8_t rx_buf0[3];
uint8_t rx_buf1[3];

main()
{
    r_main_userinit();
    /* Start user code for main. Do not edit comment generated here */
    R_Config_UART0_Start();
    R_Config_UART1_Start();

    R_Config_UART1_Receive(rx_buf1, 3);
    R_Config_UART0_Send(tx_buf0, 3);
    while(transmitend_flag_0 != 1);
    while(receiveend_flag_1 != 1);

    R_Config_UART0_Receive(rx_buf0, 3);
    R_Config_UART1_Send(tx_buf1, 3);
    while(transmitend_flag_1 != 1);
    while(receiveend_flag_0 != 1);

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

Config_UARTA0_user.c
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t transmitend_flag_0 = 0;
volatile uint8_t receiveend_flag_0 = 0;
/* End user code. Do not edit comment generated here */

void r_Config_UART0_callback_sendend(void)
{
    /* Start user code for r_Config_UART0_callback_sendend. Do not edit comment generated here */
    transmitend_flag_0 = 1;
    /* End user code. Do not edit comment generated here */
}

void r_Config_UART0_callback_receiveend(void)
{
    /* Start user code for r_Config_UART0_callback_receiveend. Do not edit comment generated here */
    receiveend_flag_0 = 1;
    /* End user code. Do not edit comment generated here */
}

Config_UARTA1_user.c
/* Start user code for global. Do not edit comment generated here */
```
void r_Config_UART1_callback_sendend(void)
{
    /* Start user code for r_Config_UART1_callback_sendend. Do not edit comment generated here */
    transmitend_flag_1 = 1;
    /* End user code. Do not edit comment generated here */
}

void r_Config_UART1_callback_receiveend(void)
{
    /* Start user code for r_Config_UART1_callback_receiveend. Do not edit comment generated here */
    receiveend_flag_1 = 1;
    /* End user code. Do not edit comment generated here */
}
### 3.2.16 Watchdog Timer

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_WDTn&gt;_Create</td>
<td>Performs initialization necessary to control the watchdog timer functions</td>
</tr>
<tr>
<td>R_&lt;Config_WDTn&gt;_Restart</td>
<td>Clears the watchdog timer counter and resumes counting</td>
</tr>
<tr>
<td>R_&lt;Config_WDTXn&gt;_Create</td>
<td>Performs initialization necessary to control the watchdog timer functions</td>
</tr>
<tr>
<td>R_&lt;Config_WDTXn&gt;_Restart</td>
<td>Clears the watchdog timer counter and resumes counting</td>
</tr>
<tr>
<td>R_&lt;Config_WDTn&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the watchdog timer functions</td>
</tr>
<tr>
<td>r_&lt;Config_WDTn&gt;_interrupt</td>
<td>Performs processing in response to the interval interrupt</td>
</tr>
<tr>
<td>R_&lt;Config_WDTXn&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the watchdog timer functions</td>
</tr>
<tr>
<td>r_&lt;Config_WDTXn&gt;_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 by `R_Systeminit` 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_WDTXn>_Create

Performs initialization necessary to control the watchdog timer functions.

Remark This API function is called by R_Systeminit function.

Syntax

void  R_<Config_WDTXn>_Create(void);

Remark X is WDT name, n is the channel number.

Argument(s) None.

Return value None.
R_<Config_WDTXn>_Restart

Clears the watchdog timer counter and resumes counting.

[Syntax]

```c
void R_<Config_WDTXn>_Restart(void);
```

[Argument(s)]
None.

[Return value]
None.
**R_<Config_WDTn>_Create_UserInit**

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

**Remark**  
This API functions 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 watchdog timer interrupt.

[Syntax]

```c
void r_<Config_WDTn>_interrupt(void);
```

Remark $n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
**R_<Config_WDTXn>_Create_UserInit**

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

**Remark**
This API function is called as the R_<Config_WDTXn>_Create callback routine.

**[Syntax]**

```c
void R_<Config_WDTXn>_Create_UserInit(void);
```

**Remark**

X is WDT name, n is the channel number.

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

**[Return value]**
None.
r_<Config_WDTXn>_interrupt

Performs processing in response to the watchdog timer interrupt.

**[Syntax]**

```c
void r_<Config_WDTXn>_interrupt(void);
```

Remark  
X is WDT name, n is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.17 Clock Divide

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

Table 3.17 API Functions: [Clock Divid]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_TAUXn_m&gt;_Create</td>
<td>Performs initialization necessary to control the clock divide</td>
</tr>
<tr>
<td></td>
<td>functions</td>
</tr>
<tr>
<td>R_&lt;Config_TAUXn_m&gt;_Start</td>
<td>Starts the count for channel m</td>
</tr>
<tr>
<td>R_&lt;Config_TAUXn_m&gt;_Stop</td>
<td>Ends the count for channel m</td>
</tr>
<tr>
<td>R_&lt;Config_TAUXn_m&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the clock</td>
</tr>
<tr>
<td></td>
<td>divide functions</td>
</tr>
<tr>
<td>r_&lt;Config_TAUXn_m&gt;_interrupt</td>
<td>Performs processing in response to the timer interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_TAUXn_m&gt;_interrupt_pek</td>
<td>Performs processing in response to the timer interrupt or PEk</td>
</tr>
</tbody>
</table>
**R_<Config_TAU<n,m>_Create**

Performs initialization necessary to control the clock divide functions.

**Remark**  This API function is called by **R_Systeminit** function.

**[Syntax]**

```c
void R_<Config_TAU<n,m>_Create(void);
```

**Remark**  TAU is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_TAUXn_m>_Start**

Starts the count for channel \( m \).

**[Syntax]**

```c
void R_<Config_TAUXn_m>_Start(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_TAUXn_m>_Stop**

Ends the count for channel \( m \).

**[Syntax]**

```c
void R_<Config_TAUXn_m>_Stop(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R<Config_TAUXn_m>_Create_UserInit**

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

**Remark**  This API function is called as the **R<Config_TAUXn_m>_Create** callback routine.

**[Syntax]**

```c
void R<Config_TAUXn_m>_Create_UserInit(void);
```

**Remark**  TAU is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
Performing processing in response to the timer interrupt for PEk.

**Syntax**

```c
void r_<Config_TAUXn_m>_interrupt_pek(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number, k is the PE number.

**Argument(s)**

None.

**Return value**

None.
r_<Config_TAUx_n_m>_interrupt

Performs processing in response to the timer interrupt.

[Syntax]

```c
void r_<Config_TAUx_n_m>_interrupt(void);
```

Remark

TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

[Argument(s)]

None.

[Return value]

None.
### 3.2.18 Data CRC

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

#### Table 3.18 API Functions: [Data CRC]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_DCRAn&gt;_Create</td>
<td>Performs initialization necessary to control the data CRC functions</td>
</tr>
<tr>
<td>R_&lt;Config_DCRAn&gt;_InitializeCRCData</td>
<td>Initializes CRC data register with the initial start value</td>
</tr>
<tr>
<td>R_&lt;Config_DCRAn&gt;_Input32bitData</td>
<td>Sets the calculation data for 32-bit width</td>
</tr>
<tr>
<td>R_&lt;Config_DCRAn&gt;_Input16bitData</td>
<td>Sets the calculation data for 16-bit width</td>
</tr>
<tr>
<td>R_&lt;Config_DCRAn&gt;_Input8bitData</td>
<td>Sets the calculation data for 8-bit width</td>
</tr>
<tr>
<td>R_&lt;Config_DCRAn&gt;_GetResult_32bitData</td>
<td>Reads the results of CRC calculation for 32-bit width</td>
</tr>
<tr>
<td>R_&lt;Config_DCRAn&gt;_GetResult_16bitData</td>
<td>Reads the results of CRC calculation for 16-bit width</td>
</tr>
<tr>
<td>R_&lt;Config_DCRAn&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the data CRC functions</td>
</tr>
<tr>
<td>R_&lt;Config_KCRCn&gt;_Create</td>
<td>Performs initialization necessary to control the data CRC functions</td>
</tr>
<tr>
<td>R_&lt;Config_KCRCn&gt;_InitializeCRCData</td>
<td>Initializes CRC data register with the initial start value</td>
</tr>
<tr>
<td>R_&lt;Config_KCRCn&gt;_Input32bitData</td>
<td>Sets the calculation data for 32-bit width</td>
</tr>
<tr>
<td>R_&lt;Config_KCRCn&gt;_Input16bitData</td>
<td>Sets the calculation data for 16-bit width</td>
</tr>
<tr>
<td>R_&lt;Config_KCRCn&gt;_Input8bitData</td>
<td>Sets the calculation data for 8-bit width</td>
</tr>
<tr>
<td>R_&lt;Config_KCRCn&gt;_GetResult_64bitData</td>
<td>Reads the results of CRC calculation for 64-bit width</td>
</tr>
<tr>
<td>R_&lt;Config_KCRCn&gt;_GetResult_32bitData</td>
<td>Reads the results of CRC calculation for 32-bit width</td>
</tr>
<tr>
<td>R_&lt;Config_KCRCn&gt;_GetResult_16bitData</td>
<td>Reads the results of CRC calculation for 16-bit width</td>
</tr>
<tr>
<td>R_&lt;Config_KCRCn&gt;_GetResult_8bitData</td>
<td>Reads the results of CRC calculation for 8-bit width</td>
</tr>
<tr>
<td>R_&lt;Config_KCRCn&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the data CRC functions</td>
</tr>
</tbody>
</table>
R_<Config_DCRAn>_Create

Performs initialization necessary to control the data CRC functions.

Remark This API function is called by R_Systeminit function.

[Syntax]

void R_<Config_DCRAn>_Create(void);

Remark $n$ is the unit number.

[Argument(s)]

None.

[Return value]

None.
**R_<Config_DCRAn>_InitializeCRCData**

Initializes CRC data register with the initial start value.

**[Syntax]**

```c
void R_<Config_DCRAn>_InitializeCRCData(uint32_t crc_data);
```

Remark  

$n$ is the unit 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 crc_data</td>
<td>DCRA initialization value</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
**R_<Config_DCRAn>_Input32bitData**

Sets the calculation data for 32-bit width.

**[Syntax]**

```c
void R_<Config_DCRAn>_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\_DCRA\_n>_Input16bitData

Sets the calculation data for 16-bit width.

**[Syntax]**

```c
void R_<Config\_DCRA\_n>_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_DCRA_n>_Input8bitData**

Sets the calculation data for 8-bit width.

**[Syntax]**

```c
void R_<Config_DCRA_n>_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 *</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_DCRA_>_GetResult_16bitData

Reads the results of CRC calculation for 16-bit width.

**[Syntax]**

```c
void R_<Config_DCRA_>_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 functions 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.
R_{<Config_KCRCn>_Create}

Performs initialization necessary to control the data CRC functions.

Remark This API function is called by R_Systeminit function.

[Syntax]

void R_{<Config_KCRCn>_Create}(void);

Remark $n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_<Config_KCRCn>_InitializeCRCData

Initializes CRC data register with the initial start value.

[Syntax]

```c
void R_<Config_KCRCn>_InitializeCRCData(uint32_t crcout0_data, uint32_t crcout1_data);
```

Remark  

\( n \) is the unit 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 crcout0_data</td>
<td>lower side of CRC initialization data</td>
</tr>
<tr>
<td>I</td>
<td>uint32_t crcout1_data</td>
<td>upper side of CRC initialization data</td>
</tr>
</tbody>
</table>

[Return value]

None.
**R\_\langle Config\_KCRCn\rangle\_Input32bitData**

Sets the calculation data for 32-bit width.

**[Syntax]**

```c
void R\_<Config\_KCRCn>_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_KCRCn>_Input16bitData**

Sets the calculation data for 16-bit width.

**[Syntax]**

```c
void _R_<Config_KCRCn>_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_{KCRC}n>_Input8bitData

Sets the calculation data for 8-bit width.

**[Syntax]**

```c
void R_<$Config_{KCRC}n>_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_KCRCn>_GetResult_64bitData**

Reads the results of CRC calculation for 64-bit width.

**[Syntax]**

```c
void R_<Config_KCRCn>_GetResult_64bitData(uint64_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>Uint64_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_KCRCn>_GetResult_32bitData**

Reads the results of CRC calculation for 32-bit width.

**[Syntax]**

```c
void R_<Config_KCRCn>_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_KCRCn>_GetResult_16bitData

Reads the results of CRC calculation for 16-bit width.

**[Syntax]**

```c
void R_<Config_KCRCn>_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\_KCRCn>_\_getResult\_8bitData

Reads the results of CRC calculation for 8-bit width.

**[Syntax]**

```c
void R\_<Config\_KCRCn>_\_getResult\_8bitData(uint8\_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>Uint8_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_KCRCn>_Create_UserInit

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

Remark  This API functions is called as the R_<Config_KCRCn>_Create callback routine.

[Syntax]

\[
\text{void } \text{R_<Config_KCRCn>_Create_UserInit(\text{void})};
\]

Remark  \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
3.2.19 Delay Count

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>R_&lt;Config_TAUXn_m&gt;_Create</code></td>
<td>Performs initialization necessary to control the delay count functions</td>
</tr>
<tr>
<td><code>R_&lt;Config_TAUXn_m&gt;_Start</code></td>
<td>Starts the count for channel <code>m</code></td>
</tr>
<tr>
<td><code>R_&lt;Config_TAUXn_m&gt;_Stop</code></td>
<td>Ends the count for channel <code>m</code></td>
</tr>
<tr>
<td><code>R_&lt;Config_TAUXn_m&gt;_Create_UserInit</code></td>
<td>Performs user-defined initialization relating to the delay count functions</td>
</tr>
<tr>
<td><code>r_&lt;Config_TAUXn_m&gt;_interrupt</code></td>
<td>Performs processing in response to the timer interrupt</td>
</tr>
<tr>
<td><code>r_&lt;Config_TAUXn_m&gt;_interrupt_peK</code></td>
<td>Performs processing in response to the timer interrupt for PEk</td>
</tr>
</tbody>
</table>
**R_{<Config\_TAUXn\_m>\_Create**

Performs initialization necessary to control the delay count functions.

**Remark**
This API function is called by \textit{R\_Systeminit} function.

**[Syntax]**

```c
void R_{<Config\_TAUXn\_m>\_Create(void);
```

**Remark**
\textit{TAUX} is TAU name (TAUB, TAUD, TAUJ), \textit{n} is the unit number, \textit{m} is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_\text{<Config\_TAUX}_n\text{\_m\_Start}

Starts the count for channel \( m \).

[Syntax]

\[
\text{void \ R\text{\_\_Config\_TAUX}_n\text{\_m\_Start}(\text{void});}
\]

Remark TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

[Argument(s)]
None.

[Return value]
None.
R$_<$Config_TAU$x_n$$_m$self>_{Stop}$

Ends the count for channel $m$.

[Syntax]

```c
void R$_<$Config_TAU$x_n$$_m$self>_{Stop}$(void);
```

Remark

TAUX is TAU name (TAUB, TAUD, TAUJ), $n$ is the unit number, $m$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_<Config_TAUXn_m>_Create_UserInit

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

Remark  This API functions is called as the R_<Config_TAUXn_m>_Create callback routine.

[Syntax]

```c
void R_<Config_TAUXn_m>_Create_UserInit(void);
```

Remark  TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

[Argument(s)]

None.

[Return value]

None.
**r_<Config_TAUXn_m>_interrupt**

Performs processing in response to the timer interrupt.

**[Syntax]**

```c
void r_<Config_TAUXn_m>_interrupt(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_<Config_TAUXn_m>_interrupt_pek

Performs processing in response to the timer interrupt for PEk.

**[Syntax]**

```c
void r_<Config_TAUXn_m>_interrupt_pek(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number, k is the PE number.

**[Argument(s)]**

None.

**[Return value]**

None.
### 3.2.20 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_&lt;Config_DMACn&gt;_Create</td>
<td>Performs initialization necessary to control the DMACn functions</td>
</tr>
<tr>
<td>R_DMACnm_Create</td>
<td>Performs initialization necessary to control the DMACn channelm functions</td>
</tr>
<tr>
<td>R_&lt;Config_DMACnm&gt;_Start</td>
<td>Enables the DMACn channelm transfer</td>
</tr>
<tr>
<td>R_&lt;Config_DMACnm&gt;_Stop</td>
<td>Disables the DMACn channelm transfer</td>
</tr>
<tr>
<td>R_&lt;Config_DMACnm&gt;_Set_SoftwareTrigger</td>
<td>Generates the CMACn channelm transfer request</td>
</tr>
<tr>
<td>R_&lt;Config_DMACnm&gt;_Suspend</td>
<td>Suspends DMACn channelm transfer</td>
</tr>
<tr>
<td>R_&lt;Config_DMACnm&gt;_Resume</td>
<td>Resumes DMACn channelm transfer</td>
</tr>
<tr>
<td>R_&lt;Config_DMACnm&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the DMACn functions</td>
</tr>
<tr>
<td>r_&lt;Config_DMACnm&gt;_dmacnm_interrupt</td>
<td>Performs processing in response to the DMACn channelm interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_DMACnm&gt;_Callback_DMA_Transfer</td>
<td>Performs processing in response to the DMACn channelm transfer error interrupt</td>
</tr>
<tr>
<td>R_&lt;Config_SDMACnm&gt;_Create</td>
<td>Performs initialization necessary to control the SDMACn channelm functions</td>
</tr>
<tr>
<td>R_&lt;Config_SDMACnm&gt;_Start</td>
<td>Enables the SDMACn channelm transfer</td>
</tr>
<tr>
<td>R_&lt;Config_SDMACnm&gt;_Stop</td>
<td>Disables the SDMACn channelm transfer</td>
</tr>
<tr>
<td>R_&lt;Config_SDMACnm&gt;_Resume</td>
<td>Resume SDMACn channelm transfer</td>
</tr>
<tr>
<td>R_&lt;Config_SDMACnm&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the SDMACn channelm functions</td>
</tr>
<tr>
<td>r_&lt;Config_SDMACnm&gt;_end_interrupt</td>
<td>Performs processing in response to the SDMACn channelm transfer end interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_SDMACnm&gt;_Callback_PEk_Address_Er</td>
<td>Performs user-defined processing relating to the SDMACn channelm address error interrupt for PEk</td>
</tr>
<tr>
<td>R_&lt;Config_SDMACnm&gt;_Set_DescriptorMemory</td>
<td>Sets descriptor memory for SDMACn channelm</td>
</tr>
</tbody>
</table>
**R_<Config_DMACn>_Create**

Performs initialization necessary to control the DMAC\textsubscript{n} functions

*Remark*  This API function is called by \textit{R_Systeminit} function.

**[Syntax]**

```c
void R_<Config_DMACn>_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 functions is called as the \texttt{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 DMAC\textit{n} channel\textit{m} transfer.

[Syntax]

```c
void R_<Config_DMACnm>_Start(void);
```

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

[Argument(s)]

None.

[Return value]

None.
Disables the DMAC\textit{n} channel\textit{m} transfer.

**[Syntax]**

```c
void R_<Config_DMACnm>_Stop(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_DMACnm>_Set_SoftwareTrigger**

Generates the DMAC\textit{n} channel\textit{m} transfer request.

**[Syntax]**

```c
void R_<Config_DMACnm>_Set_SoftwareTrigger(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_DMACnm>_Suspend

Suspends DMACn channel/m transfer.

[Syntax]

```c
void R_<Config_DMACnm>_Suspend(void);
```

Remark  

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

[Argument(s)]

None.

[Return value]

None.
**R_<Config_DMACnm>_Resume**

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

**[Syntax]**

\begin{verbatim}
void R_<Config_DMACnm>_Resume(void);
\end{verbatim}

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R\_<Config\_DMACn>\_Create\_UserInit**

Performs user-defined initialization relating to the DMAC\(n\) functions.

**Remark**  This API functions 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_\(<\text{Config\_DMAC}nm\>_\_\text{dmac}nm\_\text{interrupt}\)**

Performs processing in response to the DMAC\(n\) channel\(m\) interrupt.

**[Syntax]**

```c
void r_\(<\text{Config\_DMAC}nm\>_\_\text{dmac}nm\_\text{interrupt}(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_\(<\text{Config}_{\text{DMAC}}nm_{\text{m}}\>_\text{Callback}_{\text{DMAC}}nm_{\text{m}}_\text{Transfer}_{\text{Error}}**

Performs processing in response to the DMAC transfer error interrupt.

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

**[Syntax]**

```c
void r_\(<\text{Config}_{\text{DMAC}}nm_{\text{m}}\>_\text{Callback}_{\text{DMAC}}nm_{\text{m}}_\text{Transfer}_{\text{Error}}(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>_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>_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.
**R_<Config_SDMACnm>_Create**

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

**Remark**
This API function is called by **R_Systeminit** function.

**[Syntax]**

```c
void R_<Config_SDMACnm>_Create(void);
```

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

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

**[Return value]**
None.
**R_<Config SDMACnm>_Start**

Enables the SDMACn channel \( m \) transfer.

**[Syntax]**

```c
void R_<Config SDMACnm>_Start(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_SDMACnm>_Stop**

Disables the SDMAC$n$ channel $m$ transfer.

**[Syntax]**

```c
void R_<Config_SDMACnm>_Stop(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_SDMACnm>_Resume

Resumes SDMACn channel m transfer.

[Syntax]

```c
void R_<Config_SDMACnm>_Resume(void);
```

Remark  

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

[Argument(s)]

None.

[Return value]

None.
R_<Config_SDMAC<n>m>_Reset

Resets SDMAC<n> channel <m>.

[Syntax]

```c
void R_<Config_SDMAC<n>m>_Reset(void);
```

Remark  


\[ n \] is the unit number, \[ m \] is the channel number.

[Argument(s)]

None.

[Return value]

None.
**R_<Config_SDMACnm>_Create_UserInit**

Performs user-defined initialization relating to the SDMAC\(n\) channel\(m\) functions.

**Remark**  This API functions is called as the **R_<Config_SDMACnm>_Create** callback routine.

**[Syntax]**

```c
void R_<Config_SDMACnm>_Create_UserInit(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_<Config_SDMACnm>_end_interrupt

Performs processing in response to the SDMAC\textit{n} channel\textit{m} transfer end interrupt.

**[Syntax]**

```c
void r_<Config_SDMACnm>_interrupt(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_\(<\text{Config\_SDMACnm}>\)_Callback\_\text{PE}k\_\text{Address\_Error}**

Performs user-defined processing in response to the SDMAC address error interrupt.

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

**[Syntax]**

```c
void r_<Config_SDMACnm>_Callback_PEk_Address_Error (void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_SDMACnm>_Set_DescriptorMemory

Sets descriptor memory for DMACn channel m.

[Syntax]

void R_<Config_SDMACnm>_Set_DescriptorMemory(uint32_t * const address, uint32_t * const regValues, uint8_t count_num);

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>I</td>
<td>uint32_t * const address</td>
<td>Descriptor memory address</td>
</tr>
<tr>
<td>I</td>
<td>uint32_t * const regValues</td>
<td>Register values</td>
</tr>
<tr>
<td>I</td>
<td>uint8_t count_num</td>
<td>Register count.</td>
</tr>
</tbody>
</table>

[Return value]

None.
3.2.21 External Event Count

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

Table 3.21 API Functions: [External Event Count]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_TAUXn_m&gt;_Create</td>
<td>Performs initialization necessary to control the external event count functions</td>
</tr>
<tr>
<td>R_&lt;Config_TAUXn_m&gt;_Start</td>
<td>Starts the count for channel m</td>
</tr>
<tr>
<td>R_&lt;Config_TAUXn_m&gt;_Stop</td>
<td>Ends the count for channel m</td>
</tr>
<tr>
<td>R_&lt;Config_TAUXn_m&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the external event count functions</td>
</tr>
<tr>
<td>r_&lt;Config_TAUXn_m&gt;_interrupt</td>
<td>Performs processing in response to the timer interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_TAUXn_m&gt;_interrupt_pek</td>
<td>Performs processing in response to the timer interrupt for PEk</td>
</tr>
</tbody>
</table>
**R_<Config_TAUXn_m>_Create**

Performs initialization necessary to control the external event count functions.

**Remark** This API function is called by **R_Systeminit** function.

**[Syntax]**

```c
void R_<Config_TAUXn_m>_Create(void);
```

**Remark** TAU is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TAUXn_m>_Start

Starts the count for channel \( m \).

**[Syntax]**

```c
void R_<Config_TAUXn_m>_Start(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TAUxn_m>_Stop

Ends the count for channel \( m \).

**[Syntax]**

<table>
<thead>
<tr>
<th>void R_&lt;Config_TAUxn_m&gt;_Stop(void);</th>
</tr>
</thead>
</table>

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R\_<Config\_TAUXn\_m\>_Create\_UserInit**

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

Remark: This API function is called as the \texttt{R\_<Config\_TAUXn\_m\>_Create} callback routine.

**[Syntax]**

\begin{verbatim}
void R\_<Config\_TAUXn\_m\>_Create\_UserInit(void);
\end{verbatim}

Remark: TAUX is TAU name (TAUB, TAUD, TAUJ), \textit{n} is the unit number, \textit{m} is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r.<Config_TAUXn_m>_interrupt**

Performs processing in response to the timer interrupt.

**[Syntax]**

```c
void r.<Config_TAUXn_m>_interrupt(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r.<Config_TAU>n_m>_interrupt_pek

Performs processing in response to the timer interrupt for PEk.

[Syntax]
void r.<Config_TAU>n_m>_interrupt_pek(void);

Remark
TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number, k is the PE number.

[Argument(s)]
None.

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

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

#### Table 3.22 API Functions: [Input Period Count Detection]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_TAUXn_m&gt;_Create</td>
<td>Performs initialization necessary to control the input period count detection functions</td>
</tr>
<tr>
<td>R_&lt;Config_TAUXn_m&gt;_Start</td>
<td>Starts the count for channel m</td>
</tr>
<tr>
<td>R_&lt;Config_TAUXn_m&gt;_Stop</td>
<td>Ends the count for channel m</td>
</tr>
<tr>
<td>R_&lt;Config_TAUXn_m&gt;_Get_PulseWidth</td>
<td>Reads the input pulse width of the timer</td>
</tr>
<tr>
<td>R_&lt;Config_TAUXn_m&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the input period count detection functions</td>
</tr>
<tr>
<td>r_&lt;Config_TAUXn_m&gt;_interrupt</td>
<td>Performs processing in response to the timer interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_TAUXn_m&gt;_interrupt_pek</td>
<td>Performs processing in response to the timer interrupt for PEk</td>
</tr>
</tbody>
</table>
**R_<Config_TAUXn_m>_Create**

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

**Remark** This API function is called by `R_Systeminit` function.

**[Syntax]**

```c
void R_<Config_TAUXn_m>_Create(void);
```

**Remark** `TAUX` is TAU name (TAUB, TAUD, TAUJ), `n` is the unit number, `m` is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TAU\text{Xn}_m>_Start

Starts the count for channel $m$.

**[Syntax]**

```c
void R_<Config_TAU\text{Xn}_m>_Start(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), $n$ is the unit number, $m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TAUXn_m>_Stop

Ends the count for channel m.

[Syntax]

```c
void R_<Config_TAUXn_m>_Stop(void);
```

Remark TAU is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

[Argument(s)]

None.

[Return value]

None.
**R_<Config_TAUXn_m>_Get_PulseWidth**

Reads the input pulse width of the timer.

**Syntax**

```c
void R_<Config_TAUXn_m>_Get_PulseWidth(uint32_t * const width);
```

Remark: TAU is TAU name (TAUB, TAUD, TAUJ), 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_TAUXn_m>_Create_UserInit

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

Remark This API functions is called as the R_<Config_TAUXn_m>_Create callback routine.

[Syntax]

void R_<Config_TAUXn_m>_Create_UserInit(void);

Remark TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

[Argument(s)]

None.

[Return value]

None.
r_<Config_TAUXn_m>_interrupt

Performs processing in response to the timer interrupt.

**[Syntax]**

```c
void r_<Config_TAUXn_m>_interrupt(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_<Config_TAU Xn_m>_interrupt_pek

Performs processing in response to the timer interrupt for PEk.

**[Syntax]**

```c
void r_<Config_TAU Xn_m>_interrupt_pek(void);
```

**Remark**

TAU is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number, k is the PE number.

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.23 Input Position Detection

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

Table 3.23 API Functions: [Input Position Detection]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>R_&lt;Config_TAUXn_m&gt;_Create</code></td>
<td>Performs initialization necessary to control the input position</td>
</tr>
<tr>
<td></td>
<td>detection functions</td>
</tr>
<tr>
<td><code>R_&lt;Config_TAUXn_m&gt;_Start</code></td>
<td>Starts the count for channel m</td>
</tr>
<tr>
<td><code>R_&lt;Config_TAUXn_m&gt;_Stop</code></td>
<td>Ends the count for channel m</td>
</tr>
<tr>
<td><code>R_&lt;Config_TAUXn_m&gt;_Get_PulseWidth</code></td>
<td>Reads the input pulse width of the timer</td>
</tr>
<tr>
<td><code>R_&lt;Config_TAUXn_m&gt;_Create_UserInit</code></td>
<td>Performs user-defined initialization relating to the input position detection functions</td>
</tr>
<tr>
<td><code>r_&lt;Config_TAUXn_m&gt;_interrupt</code></td>
<td>Performs processing in response to the timer interrupt</td>
</tr>
<tr>
<td><code>r_&lt;Config_TAUXn_m&gt;_interrupt_pek</code></td>
<td>Performs processing in response to the timer interrupt for PEk</td>
</tr>
</tbody>
</table>
**R_<Config_TAUXn_m>_Create**

Performs initialization necessary to control the input position detection functions.

**Remark**  This API function is called by **R_Systeminit** function.

**[Syntax]**

```c
void R_<Config_TAUXn_m>_Create(void);
```

**Remark**  TAU is TAU name (TAUB, TAUD, TAUJ), *n* is the unit number, *m* is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_{<\text{Config}_{\text{TAUX}}_{n\_m}\>_\text{Start}}

Starts the count for channel $m$.

**[Syntax]**

```c
void R_{<\text{Config}_{\text{TAUX}}_{n\_m}\>_\text{Start}}(void);
```

Remark: TAUX is TAU name (TAUB, TAUD, TAUJ), $n$ is the unit number, $m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TAUXn_m>_Stop

Ends the count for channel m.

[Syntax]

void R_<Config_TAUXn_m>_Stop(void);

Remark

TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

[Argument(s)]

None.

[Return value]

None.
**R_{<Config\_TAUXn\_m>\_Get\_PulseWidth}**

Reads the input pulse width of the timer.

**[Syntax]**

```c
void R_{<Config\_TAUXn\_m>\_Get\_PulseWidth}(uint32_t * const width);
```

Remark: TAUX is TAU name (TAUB, TAUD, TAUJ), 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_TAUXn_m>_Create_UserInit

Performs user-defined initialization relating to the input position detection functions.
Remark This API functions is called as the R_<Config_TAUXn_m>_Create callback routine.

[Syntax]

void R_<Config_TAUXn_m>_Create_UserInit(void);

Remark TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

[Argument(s)]

None.

[Return value]

None.
**r_<Config_TAUXn_m>_interrupt**

Performs processing in response to the timer interrupt.

**[Syntax]**

```c
void r_<Config_TAUXn_m>_interrupt(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_<Config_TAUXn_m>_interrupt_pek

Performs processing in response to the timer interrupt for PEk.

[Syntax]

```c
void r_<Config_TAUXn_m>_interrupt_pek(void);
```

Remark

TAU is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number, k is the PE number.

[Argument(s)]

None.

[Return value]

None.
### Input Pulse Interval Judgment

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

**Table 3.24 API Functions: [Input Pulse Interval Judgment]**

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_TAUXn_m&gt;_Create</td>
<td>Performs initialization necessary to control the input pulse interval judgment functions.</td>
</tr>
<tr>
<td>R_&lt;Config_TAUXn_m&gt;_Start</td>
<td>Starts the count for channel m.</td>
</tr>
<tr>
<td>R_&lt;Config_TAUXn_m&gt;_Stop</td>
<td>Ends the count for channel m.</td>
</tr>
<tr>
<td>R_&lt;Config_TAUXn_m&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the input pulse interval judgment functions.</td>
</tr>
<tr>
<td>r_&lt;Config_TAUXn_m&gt;_interrupt</td>
<td>Performs processing in response to the timer interrupt.</td>
</tr>
<tr>
<td>r_&lt;Config_TAUXn_m&gt;_interrupt_pek</td>
<td>Performs processing in response to the timer interrupt for PEk</td>
</tr>
</tbody>
</table>
**R_<Config_TAUXn_m>_Create**

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

**Remark**  
This API function is called by **R_Systeminit** function.

**[Syntax]**

```c
void R_<Config_TAUXn_m>_Create(void);
```

**Remark**  
TAUX is TAU name (TAUB, TAUD, TAUJ), *n* is the unit number, *m* is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_{<Config_TAU\text{X}_n\_m>\_Start}

Starts the count for channel $m$.

**[Syntax]**

\[
\text{void R}_{<\text{Config}_\text{TAUXn}_m>\_Start(\text{void});}
\]

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), $n$ is the unit number, $m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_{<Config_TAUXn_m>_Stop}**

Ends the count for channel m.

**[Syntax]**

```c
void R_{<Config_TAUXn_m>_Stop(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_TAUXn_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_TAUXn_m>_Create callback routine.

**[Syntax]**

```c
void R_<Config_TAUXn_m>_Create_UserInit(void);
```

**Remark**
TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

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

**[Return value]**
None.
r_<Config_TAUx_n_m>_interrupt

Performs processing in response to the timer interrupt.

[Syntax]

```c
void r_<Config_TAUx_n_m>_interrupt(void);
```

Remark

TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

[Argument(s)]

None.

[Return value]

None.
r_<Config_TAUXn_m>_interrupt_pek

Performs processing in response to the timer interrupt for PEk.

**[Syntax]**

```c
void r_<Config_TAUXn_m>_interrupt_pek(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number, \( k \) is the PE number.

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.25 Input Signal Width Judgment

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

### Table 3.25 API Functions: [Input Signal Width Judgment]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_{Config_TAUx_n} Create</td>
<td>Performs initialization necessary to control the input signal width judgment functions</td>
</tr>
<tr>
<td>R_{Config_TAUx_n} Start</td>
<td>Starts the count for channel $m$</td>
</tr>
<tr>
<td>R_{Config_TAUx_n} Stop</td>
<td>Ends the count for channel $m$</td>
</tr>
<tr>
<td>R_{Config_TAUx_n} Create_UserInit</td>
<td>Performs user-defined initialization relating to the input signal width judgment functions</td>
</tr>
<tr>
<td>r_{Config_TAUx_n} interrupt</td>
<td>Performs processing in response to the timer interrupt</td>
</tr>
<tr>
<td>r_{Config_TAUx_n} interrupt_pek</td>
<td>Performs processing in response to the timer interrupt for PE$k$</td>
</tr>
</tbody>
</table>
### R_<Config_TAUXn_m>_Create

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

**Remark**  
This API function is called by R_Systeminit function.

#### [Syntax]

```c
void R_<Config_TAUXn_m>_Create(void);
```

**Remark**  
TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

#### [Argument(s)]

- None.

#### [Return value]

- None.
R_<Config_TAUxn_m>_Start

Starts the count for channel \( m \).

[Syntax]

```c
void R_<Config_TAUxn_m>_Start(void);
```

Remark

TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_<Config_TAUXn_m>_Stop

Ends the count for channel $m$.

**[Syntax]**

```c
void R_<Config_TAUXn_m>_Stop(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), $n$ is the unit number, $m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R=<Config_TAUx_n_m>_Create_UserInit

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

Remark  his API functions is called as the R=<Config_TAUx_n_m>_Create callback routine.

[Syntax]

void R=<Config_TAUx_n_m>_Create_UserInit(void);

Remark  TAUx is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

[Argument(s)]

None.

[Return value]

None.
Performs processing in response to the timer interrupt.

**Syntax**

```c
void r_<Config_TAUXn_m>_interrupt(void);
```

Remark: TAUX is TAU name (TAUB, TAUD, TAUU), \( n \) is the unit number, \( m \) is the channel number.

**Argument(s)**

None.

**Return value**

None.
r_<Config_TAUXn_m>_interrupt_pek

Performs processing in response to the timer interrupt for PEk.

[Syntax]

```c
void r_<Config_TAUXn_m>_interrupt_pek(void);
```

[Remark]

TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number, k is the PE number.

[Argument(s)]

None.

[Return value]

None.
3.2.26 Input Signal Width Measurement

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

Table 3.26 API Functions: [Input Signal Width Measurement]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_TAUXn_m&gt;_Create</td>
<td>Performs initialization necessary to control the input signal width measurement functions.</td>
</tr>
<tr>
<td>R_&lt;Config_TAUXn_m&gt;_Start</td>
<td>Starts the count for channel m.</td>
</tr>
<tr>
<td>R_&lt;Config_TAUXn_m&gt;_Stop</td>
<td>Ends the count for channel m.</td>
</tr>
<tr>
<td>R_&lt;Config_TAUXn_m&gt;_Get_PulseWidth</td>
<td>Reads the input pulse width of the timer.</td>
</tr>
<tr>
<td>R_&lt;Config_TAUXn_m&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the input signal width measurement functions.</td>
</tr>
<tr>
<td>r_&lt;Config_TAUXn_m&gt;_interrupt</td>
<td>Performs processing in response to the timer interrupt.</td>
</tr>
<tr>
<td>r_&lt;Config_TAUXn_m&gt;_interrupt_pek</td>
<td>Performs processing in response to the timer interrupt for PEk.</td>
</tr>
</tbody>
</table>
R_<Config_TAUXn_m>_Create

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

Remark This API function is called by R_Systeminit function.

[Syntax]

void R_<Config_TAUXn_m>_Create(void);

Remark TAU is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

[Argument(s)]
None.

[Return value]
None.
**R\_<Config\_TAUXn\_m\>_Start**

Starts the count for channel \( m \).

**[Syntax]**

```c
void R\_<Config\_TAUXn\_m\>_Start(void);
```

**Remark**
TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TAUXn_m>_Stop

Ends the count for channel \( m \).

**[Syntax]**

```c
void R_<Config_TAUXn_m>_Stop(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TAUXn_m>_Get_PulseWidth

Reads the input pulse width of the timer.

[Syntax]

```c
void R_<Config_TAUXn_m>_Get_PulseWidth(uint32_t * const width);
```

Remark
TAUX is TAU name (TAUB, TAUD, TAUJ), 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_TAUx_n>_Create_UserInit**

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

**Remark**  This API function is called as the R_<Config_TAUx_n>_Create callback routine.

**[Syntax]**

```c
void R_<Config_TAUx_n>_Create_UserInit(void);
```

**Remark**  TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_TAUX\text{n}_m>_interrupt**

Performs processing in response to the timer interrupt.

**[Syntax]**

```c
void r_<Config_TAUX\text{n}_m>_interrupt(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
Performs processing in response to the timer interrupt for PEk.

**[Syntax]**
```c
void r_<Config_TAUXn_m>_interrupt_pek(void);
```

**Remark**
TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number, k is the PE number.

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

**[Return value]**
None.
3.2.27 Key Interrupt Function

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

Table 3.27 API Functions: [Key Interrupt Function]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R.&lt;Config_KEY&gt;_Create</td>
<td>Performs initialization necessary to control the key interrupt functions</td>
</tr>
<tr>
<td>R.&lt;Config_KEY&gt;_Start</td>
<td>Enables the acceptance of the key interrupt</td>
</tr>
<tr>
<td>R.&lt;Config_KEY&gt;_Stop</td>
<td>Disables the acceptance of the key interrupt</td>
</tr>
<tr>
<td>R.&lt;Config_KEY&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the key interrupt functions</td>
</tr>
<tr>
<td>r.&lt;Config_KEY&gt;_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 by R_Systeminit 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]**

```c
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 functions is called as the R_<Config_KEY>_Create callback routine.

[Syntax]

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.
3.2.28 One Pulse Output

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

Table 3.28 API Functions: [One Pulse Output]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_TAUxn_m&gt;_Create</td>
<td>Performs initialization necessary to control the one pulse output functions.</td>
</tr>
<tr>
<td>R_&lt;Config_TAUxn_m&gt;_Start</td>
<td>Starts the count for channel ( m ).</td>
</tr>
<tr>
<td>R_&lt;Config_TAUxn_m&gt;_Stop</td>
<td>Ends the count for channel ( m ).</td>
</tr>
<tr>
<td>R_&lt;Config_TAUxn_m&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the one pulse output functions.</td>
</tr>
<tr>
<td>r_&lt;Config_TAUxn_m&gt;_interrupt</td>
<td>Performs processing in response to the timer interrupt.</td>
</tr>
<tr>
<td>r_&lt;Config_TAUxn_m&gt;_interrupt_pek</td>
<td>Performs processing in response to the timer interrupt for PEk.</td>
</tr>
</tbody>
</table>
**R_\textless Config\_TAUXn\_m\textgreater _Create**

Performs initialization necessary to control the one pulse output functions.

**Remark** This API function is called by \texttt{R\_Systeminit} function.

**[Syntax]**

```c
void \texttt{R\_\textless Config\_TAUXn\_m\textgreater \_Create(void);}
```

**Remark** TAU is TAU name (TAUB, TAUD, TAUJ), \texttt{n} is the unit number, \texttt{m} is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_TAU\text{n}_m>_Start**

Starts the count for channel \( m \).

**[Syntax]**

```c
void R_<Config_TAU\text{n}_m>_Start(void);
```

**Remark**

TAU is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<_Config_TAUXn_m>_Stop

Ends the count for channel \( m \).

**[Syntax]**

```c
void R_<_Config_TAUXn_m>_Stop(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TAUXn_m>_Create_UserInit

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

Remark This API functions is called as the R_<Config_TAUXn_m>_Create callback routine.

**[Syntax]**

```c
void R_<Config_TAUXn_m>_Create_UserInit(void);
```

Remark TAU is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_<Config_TAU\(n\)_\(m\)>_interrupt

Performs processing in response to the timer interrupt.

**[Syntax]**

```c
void r_<Config_TAU\(n\)_\(m\)>_interrupt(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), \(n\) is the unit number, \(m\) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_TAUXn_m>_interrupt_pek**

Performs processing in response to the timer interrupt for PE<sub>k</sub>.

**[Syntax]**

```c
void r_<Config_TAUXn_m>_interrupt_pek(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), <i>n</i> is the unit number, <i>m</i> is the channel number, <i>k</i> is the PE number.

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.29 One-shot Pulse Output

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

Table 3.29 API Functions: [One-shot Pulse Output]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_TAUXn&gt;_Create</td>
<td>Performs initialization necessary to control the one-shot pulse output functions</td>
</tr>
<tr>
<td>R_&lt;Config_TAUXn&gt;_Start</td>
<td>Starts the count</td>
</tr>
<tr>
<td>R_&lt;Config_TAUXn&gt;_Stop</td>
<td>Ends the count</td>
</tr>
<tr>
<td>R_&lt;Config_TAUXn&gt;_SoftwareTriggerOn</td>
<td>Generates the TAU channel start trigger by software</td>
</tr>
<tr>
<td>R_&lt;Config_TAUXn&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the one-shot pulse output functions</td>
</tr>
<tr>
<td>r_&lt;Config_TAUXn&gt;_channelm_interrupt</td>
<td>Performs processing in response to the timer interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_TAUXn&gt;_channelm_interrupt_pek</td>
<td>Performs processing in response to the timer interrupt for PEk</td>
</tr>
</tbody>
</table>
**R_<Config_TAUxn>_Create**

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

**Remark**  This API function is called by `R_Systeminit` function.

**[Syntax]**

```c
void R_<Config_TAUxn>_Create(void);
```

**Remark**  TAU is TAU name (TAUB, TAUD, TAUJ), `n` is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TAUXn>_Start

Starts the count.

[Syntax]

```c
void R_<Config_TAUXn>_Start(void);
```

Remark:
TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number.

[Argument(s)]

None.

[Return value]

None.
R_<Config_TAU\text{Xn}>._Stop

Ends the count.

**[Syntax]**

```c
void R_<Config_TAU\text{Xn}>._Stop(void);
```

**Remark**

`TAUX` is TAU name (TAUB, TAUD, TAUJ), `n` is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_TAU>\_SoftwareTriggerOn**

Generates the TAU channel start trigger by software.

**[Syntax]**

```c
void R_<Config_TAU>\_SoftwareTriggerOn(void);
```

Remark: TAU is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_TAU<Xn>_Create_UserInit**

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

**Remark**
This API function is called as the **R_<Config_TAU<Xn>_Create** callback routine.

**[Syntax]**

```c
void R_<Config_TAU<Xn>_Create_UserInit(void);
```

**Remark**
TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number.

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

**[Return value]**
None.
r_<Config_TAUXn>_channelm_interrupt_pek

Performs processing in response to the timer interrupt for PEk

[Syntax]

void r_<Config_TAUXn>_channelm_interrupt_pek(void);

Remark
TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number, k is the PE number.

[Argument(s)]
None.

[Return value]
None.
**r_<Config_TAUxn>_channelm_interrupt**

Performs processing in response to the timer interrupt.

**[Syntax]**

```c
void r_<Config_TAUxn>_channelm_interrupt(void);
```

**Remark**

TAU is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
### 3.2.30 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><code>R_&lt;Config_TAUXn_m&gt;_Create</code></td>
<td>Performs initialization necessary to control the overflow interrupt output (input period count detecting) functions</td>
</tr>
<tr>
<td><code>R_&lt;Config_TAUXn_m&gt;_Start</code></td>
<td>Starts the count for channel <code>m</code></td>
</tr>
<tr>
<td><code>R_&lt;Config_TAUXn_m&gt;_Stop</code></td>
<td>Ends the count for channel <code>m</code></td>
</tr>
<tr>
<td><code>R_&lt;Config_TAUXn_m&gt;_Create_UserInit</code></td>
<td>Performs user-defined initialization relating to the overflow interrupt output (input period count detecting) functions</td>
</tr>
<tr>
<td><code>r_&lt;Config_TAUXn_m&gt;_interrupt</code></td>
<td>Performs processing in response to the timer interrupt</td>
</tr>
<tr>
<td><code>r_&lt;Config_TAUXn_m&gt;_interrupt_pek</code></td>
<td>Performs processing in response to the timer interrupt for PEk</td>
</tr>
</tbody>
</table>
**R_<Config_TAUXn_m>_Create**

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

**Remark** This API function is called by `R_Systeminit` function.

**[Syntax]**

```c
void R_<Config_TAUXn_m>_Create(void);
```

**Remark** TAU is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TAUxn_m>_Start

Starts the count for channel m.

[Syntax]

```c
void R_<Config_TAUxn_m>_Start(void);
```

Remark

TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_<Config_TAU_Xn_m>_Stop

Ends the count for channel \( m \).

**[Syntax]**

```c
void R_<Config_TAU_Xn_m>_Stop(void);
```

Remark: TAU is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_TAUXn_m>_Create_UserInit**

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_TAUXn_m>_Create callback routine.

**[Syntax]**

```c
void R_<Config_TAUXn_m>_Create_UserInit(void);
```

**Remark**  TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
### r_\langle Config\_TAUXn\_m\rangle\_interrupt

Performs processing in response to the timer interrupt.

#### Syntax

```c
void r_\langle Config\_TAUXn\_m\rangle\_interrupt(void);
```

**Remark**  
TAUX is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

#### Argument(s)

None.

#### Return value

None.
r_.<Config_TAUxn_m>_interrupt_pek

Performs processing in response to the timer interrupt for PEk.

**[Syntax]**

```c
void r_.<Config_TAUxn_m>_interrupt_pek(void);
```

**Remark**

TAU is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number, k is the PE number.

**[Argument(s)]**

None.

**[Return value]**

None.
### 3.2.31 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 3.31 API Functions: [Overflow Interrupt Output (Width measurement)]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_TAUXn_m&gt;_Create</td>
<td>Performs initialization necessary to control the overflow interrupt output (width measurement) functions</td>
</tr>
<tr>
<td>R_&lt;Config_TAUXn_m&gt;_Start</td>
<td>Starts the count for channel m</td>
</tr>
<tr>
<td>R_&lt;Config_TAUXn_m&gt;_Stop</td>
<td>Ends the count for channel m</td>
</tr>
<tr>
<td>R_&lt;Config_TAUXn_m&gt;_Get_PulseWidth</td>
<td>Reads the input pulse width of the timer</td>
</tr>
<tr>
<td>R_&lt;Config_TAUXn_m&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the overflow interrupt output (width measurement) functions</td>
</tr>
<tr>
<td>r_&lt;Config_TAUXn_m&gt;_interrupt</td>
<td>Performs processing in response to the timer interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_TAUXn_m&gt;_interrupt_pek</td>
<td>Performs processing in response to the timer interrupt for PEk</td>
</tr>
</tbody>
</table>
**R_<Config_TAUx_n_m>_Create**

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

**Remark**  This API function is called by `R_Systeminit` function.

**[Syntax]**

```c
void R_<Config_TAUx_n_m>_Create(void);
```

**Remark**  TAU is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TAUXn_m>_Start

Starts the count for channel $m$.

**[Syntax]**

```c
void R_<Config_TAUXn_m>_Start(void);
```

Remark

TAUX is TAU name (TAUB, TAUD, TAUJ), $n$ is the unit number, $m$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TAU_n_m>_Stop

Ends the count for channel m.

[Syntax]

```c
void R_<Config_TAU_n_m>_Stop(void);
```

Remark

TAU is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_<Config_TAUXn_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_TAUXn_m>_Create callback routine.

[Syntax]

```c
void R_<Config_TAUXn_m>_Create_UserInit(void);
```

Remark  TAUX is TAU name (TAUB, TAUD, TAUJ), n is the unit number, m is the channel number.

[Argument(s)]

None.

[Return value]

None.
**r_<Config_TAU>n_m>_interrupt**

Performs processing in response to the timer interrupt.

**[Syntax]**

```c
void r_<Config_TAU>n_m>_interrupt(void);
```

**Remark**

TAU is TAU name (TAUB, TAUD, TAUJ), \( n \) is the unit number, \( m \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_\text{Config\_TAUXn\_m\_interrupt\_pek}**

Performs processing in response to the timer interrupt for PE\text{k}.

**[Syntax]**

```c
void r_\text{Config\_TAUXn\_m\_interrupt\_pek}(void);
```

**Remark**

TAUX is TAU name (TAUB, TAUD, TAUJ), \text{n} is the unit number, \text{m} is the channel number, \text{k} is the PE number.

**[Argument(s)]**

None.

**[Return value]**

None.
### 3.2.32 IIC Master Mode

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

**Table 3.32 API Functions: [IIC Master Mode]**

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_RIICn&gt;_Create</td>
<td>Performs initialization necessary to control the IIC master mode functions</td>
</tr>
<tr>
<td>R_&lt;Config_RIICn&gt;_Start</td>
<td>Enables IIC</td>
</tr>
<tr>
<td>R_&lt;Config_RIICn&gt;_Stop</td>
<td>Disables IIC</td>
</tr>
<tr>
<td>R_&lt;Config_RIICn&gt;_Master_Send</td>
<td>Starts master transmission</td>
</tr>
<tr>
<td>R_&lt;Config_RIICn&gt;_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_&lt;Config_RIICn&gt;_Master_Receive</td>
<td>Starts master reception</td>
</tr>
<tr>
<td>R_&lt;Config_RIICn&gt;_StartCondition</td>
<td>Issues a start condition</td>
</tr>
<tr>
<td>R_&lt;Config_RIICn&gt;_StopCondition</td>
<td>Issues a stop condition</td>
</tr>
<tr>
<td>R_&lt;Config_RIICn&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the IIC master mode functions</td>
</tr>
<tr>
<td>r_&lt;Config_RIICn&gt;_error_interrupt</td>
<td>Executes processing in response to communication error interrupts or communication event generation interrupts</td>
</tr>
<tr>
<td>r_&lt;Config_RIICn&gt;_receive_interrupt</td>
<td>Executes processing in response to receive data full interrupts</td>
</tr>
<tr>
<td>r_&lt;Config_RIICn&gt;_transmit_interrupt</td>
<td>Executes processing in response to transmit data empty interrupts</td>
</tr>
<tr>
<td>r_&lt;Config_RIICn&gt;_transmitend_interrupt</td>
<td>Executes processing in response to transmit end interrupts</td>
</tr>
<tr>
<td>r_&lt;Config_RIICn&gt;_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_&lt;Config_RIICn&gt;_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_&lt;Config_RIICn&gt;_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_RIICh>_Create**

Performs initialization necessary to control the IIC master mode functions.

**Remark**  
This API function is called by **R_Systeminit** function.

**[Syntax]**

```c
void R_<Config_RIICh>_Create(void);
```

**Remark**  

$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_RII<n>_Start

Enables IIC.

[Syntax]

```c
void R_<Config_RII<n>_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>_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]**

```
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  <code>adr</code></td>
<td>Slave address</td>
</tr>
<tr>
<td>I</td>
<td>uint8_t * const  <code>tx_buf</code></td>
<td>Pointer to the buffer where the data to be transmitted are stored</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t  <code>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 <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 \texttt{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 \texttt{tx_buf} the number of times specified by the argument \texttt{tx_num}.

Remark2 This API function internally calls \texttt{R_<Config_RIICn>_StartCondition} to start master transmission.

Remark3 \texttt{r_<Config_RIICn>_transmitend_interrupt} does not issue a stop condition to stop master transmission.

Remark4 Calling \texttt{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>uint16_t \texttt{adr}</td>
<td>Slave address</td>
</tr>
<tr>
<td>I</td>
<td>uint8_t * const \texttt{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 \texttt{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 \texttt{adr} 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 `adr` to slave devices, and then repeats the master 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**  This API function internally calls `R_<Config_RIICn>_StartCondition` to start master reception.

**Remark3**  A stop condition is issued in `r_<Config_RIICn>_receive_interrupt` to stop master reception.

**Remark4**  Calling `R_<Config_RIICn>_Start` is required before this API function is called to execute master reception.

**[Syntax]**

```c
MD_STATUS R_<Config_RIICn>_Master_Receive(uint16_t adr, uint8_t * const rx_buf, uint16_t rx_num);
```

Remark  `n` is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>uint16_t <code>adr</code></td>
<td>Slave address</td>
</tr>
<tr>
<td>O</td>
<td>uint8_t * const <code>rx_buf</code></td>
<td>Pointer to the buffer where the received data are to be stored</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t <code>rx_num</code></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>
<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>_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 master mode functions.

**Remark**  
This API functions 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.
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_RIICn>_receive_interrupt

Executes processing in response to receive data full interrupts.

**[Syntax]**

```c
void r_<Config_RIICn>_receive_interrupt(void);
```

Remark  
$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_<Config_RIICn>_transmit_interrupt

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>_transmitend_interrupt

Executes processing in response to transmit end interrupts.

[Syntax]

```
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 master transmission among the sources of communication error interrupts or communication event generation interrupts.

**Remark1**  This API function is called by 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 by 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 by 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 by *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.
Usage example

This is an example for RIIC0 as master to send data and RIIC1 as slave to receive data:
(Blue code is user code.)

```
r_cg_main.c
/* Start user code for global. Do not edit comment generated here */
uint8_t tx_buf[6] = {0xAA, 0xFF, 0x00, 0x55, 0xCC, 0x33};
volatile uint8_t rci0_master_sendend_flag = 0x00U;
volatile uint8_t rci1_slave_receiveend_flag = 0x00U;
/* End user code. Do not edit comment generated here */

void main(void)
{
    r_main_userinit();
    /* Start user code for main. Do not edit comment generated here */
    R_Config_RIIC1_Start();
    R_Config_RIIC0_Start();
    R_Config_RIIC1_Slave_Receive(rx_buf, 6);
    R_Config_RIIC0_Master_Send(0x00, rx_buf, 6);
    while(rci0_master_sendend_flag == 0 || rci0_slave_receiveend_flag == 0);
    while(1);
    /* End user code. Do not edit comment generated here */
}
```

```
Config_RIIC0_user.c
/* Start user code for global. Do not edit comment generated here */
extern volatile uint8_t rci0_master_sendend_flag;
/* End user code. Do not edit comment generated here */

void r_Config_RIIC0_callback_transmitend(void)
{
    /* Start user code for r_Config_RIIC0_callback_transmitend. Do not edit comment generated here */
    rci0_master_sendend_flag = 1;
    /* End user code. Do not edit comment generated here */
}
```

```
Config_RIIC1_user.c
/* Start user code for global. Do not edit comment generated here */
extern volatile uint8_t rci1_slave_receiveend_flag;
/* End user code. Do not edit comment generated here */

void r_Config_RIIC1_callback_receiveend(void)
{
    /* Start user code for r_Config_RIIC1_callback_receiveend. Do not edit comment generated here */
    rci1_slave_receiveend_flag = 1;
    /* End user code. Do not edit comment generated here */
}
```
3.2.33 IIC Slave Mode

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

Table 3.33 API Functions: [IIC Slave Mode]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>R_&lt;Config_RIICn&gt;_Create</code></td>
<td>Performs initialization necessary to control the IIC slave mode functions</td>
</tr>
<tr>
<td><code>R_&lt;Config_RIICn&gt;_Start</code></td>
<td>Enables IIC</td>
</tr>
<tr>
<td><code>R_&lt;Config_RIICn&gt;_Stop</code></td>
<td>Disables IIC</td>
</tr>
<tr>
<td><code>R_&lt;Config_RIICn&gt;_Slave_Send</code></td>
<td>Starts slave transmission</td>
</tr>
<tr>
<td><code>R_&lt;Config_RIICn&gt;_Slave_Receive</code></td>
<td>Starts slave reception</td>
</tr>
<tr>
<td><code>R_&lt;Config_RIICn&gt;_StartCondition</code></td>
<td>Issues a start condition</td>
</tr>
<tr>
<td><code>R_&lt;Config_RIICn&gt;_StopCondition</code></td>
<td>Issues a stop condition</td>
</tr>
<tr>
<td><code>R_&lt;Config_RIICn&gt;_Create_UserInit</code></td>
<td>Performs user-defined initialization relating to the IIC slave mode functions</td>
</tr>
<tr>
<td><code>r_&lt;Config_RIICn&gt;_error_interrupt</code></td>
<td>Executes processing in response to communication error interrupts or communication event generation interrupts</td>
</tr>
<tr>
<td><code>r_&lt;Config_RIICn&gt;_receive_interrupt</code></td>
<td>Executes processing in response to receive data full interrupts</td>
</tr>
<tr>
<td><code>r_&lt;Config_RIICn&gt;_transmit_interrupt</code></td>
<td>Executes processing in response to transmit data empty interrupts</td>
</tr>
<tr>
<td><code>r_&lt;Config_RIICn&gt;_transmitend_interrupt</code></td>
<td>Executes processing in response to transmit end interrupts</td>
</tr>
<tr>
<td><code>r_&lt;Config_RIICn&gt;_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_&lt;Config_RIICn&gt;_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 interrupt</td>
</tr>
<tr>
<td><code>r_&lt;Config_RIICn&gt;_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_{<\text{Config}\_\text{RIIC}n>\_Create**}

Performs initialization necessary to control the IIC slave mode functions.

Remark  This API function is called by R\_Systeminit function.

**[Syntax]**

```c
void R_{<\text{Config}\_\text{RIIC}n>\_Create(void);
```

Remark  \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R\_<Config\_RIIC>n>_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]
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

```c
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 \texttt{r_<Config_RIICn>_error_interrupt} is called.

[Syntax]

\begin{verbatim}
void R_<Config_RIICn>_StartCondition(void);
\end{verbatim}

Remark \texttt{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 \texttt{r_<Config_RIICn>_error_interrupt} is called.

**[Syntax]**

\begin{verbatim}
void R_<Config_RIICn>_StopCondition(void);
\end{verbatim}

**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.
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_RIICn>_receive_interrupt

Executes processing in response to receive data full interrupts.

[Syntax]

```c
void r_<Config_RIICn>_receive_interrupt(void);
```

Remark       \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
r_<Config_RII<n>>_transmit_interrupt

Executes processing in response to transmit data empty interrupts.

[Syntax]

```c
void r_<Config_RII<n>>_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 slave transmission among the sources of communication error interrupts or communication event generation interrupts.

Remark1 This API function is called by 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 by *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 by 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.
Usage example

Refer to IIC master mode Usage example.
3.2.34 SCI3 Clock Synchronous Mode

Below is a list of API functions output by the Code Generator for SCI3 Clock Synchronous Mode use.

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_SCI3n&gt;_Create</td>
<td>Performs initialization necessary to control the SCI3n Clock Synchronous Mode functions</td>
</tr>
<tr>
<td>R_&lt;Config_SCI3n&gt;_Start</td>
<td>Enables SCI3n</td>
</tr>
<tr>
<td>R_&lt;Config_SCI3n&gt;_Stop</td>
<td>Disables SCI3n</td>
</tr>
<tr>
<td>R_&lt;Config_SCI3n&gt;_Send</td>
<td>Starts SCI3n data transmission</td>
</tr>
<tr>
<td>R_&lt;Config_SCI3n&gt;_Receive</td>
<td>Starts SCI3n data reception</td>
</tr>
<tr>
<td>R_&lt;Config_SCI3n&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the SCI3n Clock Synchronous Mode functions</td>
</tr>
<tr>
<td>r_&lt;Config_SCI3n&gt;_interrupt_send</td>
<td>Performs processing in response to the SCI3n Clock Synchronous Mode communication interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_SCI3n&gt;_interrupt_receive</td>
<td>Performs processing in response to the SCI3n reception interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_SCI3n&gt;_interrupt_error</td>
<td>Performs processing in response to the SCI3n error interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_SCI3n&gt;_interrupt_sendend</td>
<td>Performs processing in response to the SCI3n send end interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_SCI3n&gt;_callback_sendend</td>
<td>Performs processing in response to the SCI3n communication interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_SCI3n&gt;_callback_receiveend</td>
<td>Performs processing in response to the SCI3n reception interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_SCI3n&gt;_callback_error</td>
<td>Performs processing in response to the SCI3n error interrupt</td>
</tr>
</tbody>
</table>
**R_<Config_SCI3n>_Create**

Performs initialization necessary to control the SCI3n Clock Synchronous Mode functions.

**Remark**  This API function is called by `R_Systeminit` function.

**[Syntax]**

```c
void R_<Config_SCI3n>_Create(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_SCI3n>_Start

Enables SCI3n.

[Syntax]

```c
void R_<Config_SCI3n>_Start(void);
```

Remark  $n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
R_<Config_SCI3n>_Stop

Disables SCI3n.

[Syntax]

```c
void R_<Config_SCI3n>_Stop(void);
```

Remark  
  n is the channel number.

[Argument(s)]  
  None.

[Return value]  
  None.
R_<Config_SC13n>_Send

Starts SCI3 data transmission.

Remark1 This API function repeats the SCI3 transmission from the buffer specified in argument tx_buf the number of times specified in argument tx_num.

Remark2 R_<Config_SC13n>_Start must be called before this API function is called.

[Syntax]

```
MD_STATUS R_<Config_SC13n>_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>
</tbody>
</table>
**R_\texttt{<Config\_SCI3n\_Receive}}**

Starts SCI3n data reception.

**Remark1**  This API function performs SCI3 reception the number of times specified by the argument \( \text{rx\_num} \), and stores the data in the buffer specified by the argument \( \text{rx\_buf} \).

**Remark2**  Starts after this API function is called, and \( \text{R\_<Config\_SCI3n\_Start}} \) is then called.

**[Syntax]**

```c
MD\_STATUS \_R\_<Config\_SCI3n\_Receive\((uint8\_t\ast const \text{rx\_buf}, uint16\_t \text{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 \text{rx_buf}</td>
<td>Pointer to a buffer to store the received data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t \text{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_SCI3n>_Create_UserInit

Performs user-defined initialization relating to the SCI3n Clock Synchronous Mode functions.

Remark This API function is called as the R_<Config_SCI3n>_Create callback routine.

[Syntax]

```c
void R_<Config_SCI3n>_Create_UserInit(void);
```

Remark \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
**r_<Config_SCI3n>_interrupt_send**

Performs processing in response to the SCI3n communication interrupt.

**[Syntax]**

```c
void r_<Config_SCI3n>_interrupt_send(void);
```

Remark  
\( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_<Config_SCI3n>_interrupt_receive

Performs processing in response to the SCI3n reception interrupt.

[Syntax]

```c
void r_<Config_SCI3n>_interrupt_receive(void);
```

Remark  
\( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
**r_<Config_SCI3>n>_interrupt_error**

Performs processing in response to the SCI3\(n\) error interrupt.

**[Syntax]**

```c
void r_<Config_SCI3>n>_interrupt_error(void);
```

Remark \(n\) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_SCI3n>_interrupt_sendend**

Performs processing in response to the SCI3n send end interrupt.

**[Syntax]**

```c
void r_<Config_SCI3n>_interrupt_sendend(void);
```

Remark  
*n* is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_SCI3n>_callback_sendend**

Performs processing in response to the SCI3 communication interrupt.

**Remark**  
This API function is called as the callback routine of interrupt process `r_<Config_SCI3n>_interrupt_send` corresponding to the SCI3 communication interrupt.

**[Syntax]**

```c
void r_<Config_SCI3n>_callback_sendend(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_SCI3n>_callback_receiveend**

Performs processing in response to the SCI3n reception interrupt.

**Remark**
This API function is called as the callback routine of interrupt process
r_<Config_SCI3n>_interrupt_receive corresponding to the SCI3 error interrupt.

**[Syntax]**

```c
void r_<Config_SCI3n>_callback_receiveend(void);
```

**Remark**

n is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
### r_<Config_SCI3n>_callback_error

**Performs processing in response to the SCI3n error interrupt.**

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

r_<Config_SCI3n>_interrupt_error corresponding to the SCI3 error interrupt.

**[Syntax]**

```c
void r_<Config_SCI3n>_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 err_type</td>
<td>Trigger for SCI3 error interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0xxxxxxxB: Transmit Data Register Empty</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x1xxxxxxB: Receive Data Register Full</td>
</tr>
<tr>
<td></td>
<td></td>
<td>xx1xxxxxB: Overrun Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>xxx1xxxxxB: Framing Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>xxxx1xxxxB: Parity Error</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
Usage example

This is an example for SCI30 to receive and send data in clock synchronous mode:
(Blue code is user code.)

```c
/* Start user code for global. Do not edit comment generated here */

uint8_t tx_buf0[6] = {0xAA, 0xFF, 0x00, 0x55, 0xCC, 0x33};
uint8_t tx_buf1[6] = {0xAA, 0xFF, 0x00, 0x55, 0xCC, 0x33};
uint8_t rx_buf0[6] = {0x88, 0x00, 0x00, 0x00, 0x00, 0x88};
uint8_t rx_buf1[6] = {0x88, 0x00, 0x00, 0x00, 0x00, 0x88};
/* End user code. Do not edit comment generated here */

void main(void)
{
    r_main_userinit();
    /* Start user code for main. Do not edit comment generated here */
    R_Config_SCI30_Start();

    R_Config_SCI30_Receive(rx_buf0, 6);
    R_Config_SCI30_Send(tx_buf0, 6);
    while(receiveend_flag != 1);
    transmitend_flag = 0;
    while(transmitend_flag != 1);
    transmitend_flag = 0;
    R_Config_SCI30_Receive(rx_buf1, 6);
    R_Config_SCI30_Send(tx_buf1, 6);
    while(receiveend_flag != 1);
    transmitend_flag = 0;
    while(transmitend_flag != 1);
    transmitend_flag = 0;
    R_Config_SCI30_Stop();
    while(1);
    /* End user code. Do not edit comment generated here */
}

Config_SCI30_user.c
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t transmitend_flag = 0;
volatile uint8_t receiveend_flag = 0;
/* End user code. Do not edit comment generated here */

void r_Config_SCI30_callback_sendend(void)
{
    /* Start user code for r_Config_SCI30_callback_sendend. Do not edit comment generated here */
    transmitend_flag = 1;
    /* End user code. Do not edit comment generated here */
}

void r_Config_SCI30_callback_receiveend(void)
{
    /* Start user code for r_Config_SCI30_callback_receiveend. Do not edit comment generated here */
    receiveend_flag = 1;
    /* End user code. Do not edit comment generated here */
}
```
### 3.2.35 SCI3 Asynchronous Mode

Below is a list of API functions output by the Code Generator for SCI3 Asynchronous Mode use.

**Table 3.35 API Functions: [SCI3 Async]**

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_SCI3n&gt;_Create</td>
<td>Performs initialization necessary to control the SCI3n Asynchronous Mode functions</td>
</tr>
<tr>
<td>R_&lt;Config_SCI3n&gt;_Start</td>
<td>Enables SCI3n</td>
</tr>
<tr>
<td>R_&lt;Config_SCI3n&gt;_Stop</td>
<td>Disables SCI3n</td>
</tr>
<tr>
<td>R_&lt;Config_SCI3n&gt;_Send</td>
<td>Starts SCI3n data transmission</td>
</tr>
<tr>
<td>R_&lt;Config_SCI3n&gt;_Receive</td>
<td>Starts SCI3n data reception</td>
</tr>
<tr>
<td>R_&lt;Config_SCI3n&gt;_Multiprocessor_Send</td>
<td>Starts SCI3n multiprocessor data transmission</td>
</tr>
<tr>
<td>R_&lt;Config_SCI3n&gt;_Multiprocessor_Receive</td>
<td>Starts SCI3n multiprocessor data reception</td>
</tr>
<tr>
<td>R_&lt;Config_SCI3n&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the SCI3n Asynchronous Mode functions</td>
</tr>
<tr>
<td>r_&lt;Config_SCI3n&gt;_interrupt_send</td>
<td>Performs processing in response to the SCI3n Asynchronous Mode communication interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_SCI3n&gt;_interrupt_receive</td>
<td>Performs processing in response to the SCI3n reception interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_SCI3n&gt;_interrupt_error</td>
<td>Performs processing in response to the SCI3n error interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_SCI3n&gt;_interrupt_sendend</td>
<td>Performs processing in response to the SCI3n send end interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_SCI3n&gt;_callback_sendend</td>
<td>Performs processing in response to the SCI3n communication interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_SCI3n&gt;_callback_receiveend</td>
<td>Performs processing in response to the SCI3n reception interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_SCI3n&gt;_callback_error</td>
<td>Performs processing in response to the SCI3n error interrupt</td>
</tr>
</tbody>
</table>
R_<Config_SCI3n>_Create

Performs initialization necessary to control the SCI3n Asynchronous Mode functions.

Remark  This API function is called by R_Systeminit function.

[Syntax]

```c
void R_<Config_SCI3n>_Create(void);
```

Remark  \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
**R_<Config_SCI3n>_Start**

Enables SCI3n.

**[Syntax]**

```c
void R_<Config_SCI3n>_Start(void);
```

Remark    

\(n\) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_SCI3n>_Stop

Disables SCI3n.

[Syntax]
```c
void R_<Config_SCI3n>_Stop(void);
```

Remark

\( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
**R_<Config_SCI3n>_Send**

Starts SCI3 data transmission.

**Remark1** This API function repeats the SCI3 transmission from the buffer specified in argument `tx_buf` the number of times specified in argument `tx_num`.

**Remark2** `R_<Config_SCI3n>_Start` must be called before this API function is called.

**[Syntax]**

```c
MD_STATUS R_<Config_SCI3n>_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>
</tbody>
</table>
### R_\(<\text{Config}_{-}\text{SCI}3n>\)_Receive

Starts SCI3n data reception.

**Remark1** This API function performs SCI3 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_\(<\text{Config}_{-}\text{SCI}3n>\)_Start` is then called.

#### [Syntax]

```
MD_STATUS R_\(<\text{Config}_{-}\text{SCI}3n>\)_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_SCI3n>_Multiprocessor_Send

Starts SCI3n multiprocessor data transmission.

Remark1  This API function repeats the SCI3 transmission from the buffer specified in argument tx_buf the number of times specified in argument tx_num.

Remark2  R_<Config_SCI3n>_Start must be called before this API function is called.

[Syntax]

```c
MD_STATUS R_<Config_SCI3n>_Multiprocessor_Send (uint8_t * const tx_buf, uint16_t tx_num, uint8_t tx_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>uint8_t * const tx_buf</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t tx_num</td>
<td>Total amount of data to send</td>
</tr>
<tr>
<td>I</td>
<td>uint16_t tx_id</td>
<td>Transmit unique ID 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 argument specification</td>
</tr>
</tbody>
</table>
R_<Config_SCI3n>_Multiprocessor_Receive

Starts SCI3n multiprocessor data reception.

Remark1  This API function performs SCI3 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_SCI3n>_Start is then called.

[Syntax]

```c
MD_STATUS R_<Config_SCI3n>_Multiprocessor_Receive(uint8_t * const rx.buf, uint16_t rx_num, uint8_t rx_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>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>
<tr>
<td>I</td>
<td>uint16_t rx_id</td>
<td>Receive unique ID 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 argument specification</td>
</tr>
</tbody>
</table>
**R_</Config_SCI3n>_Create_UserInit**

Performs user-defined initialization relating to the SCI3n Asynchronous Mode functions.

**Remark**  This API function is called as the R_</Config_SCI3n>_Create callback routine.

**[Syntax]**

```c
void R_</Config_SCI3n>_Create_UserInit(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_<Config_SCI3n>_interrupt_send

Performs processing in response to the SCI3n communication interrupt.

[Syntax]

```c
void r_<Config_SCI3n>_interrupt_send(void);
```

Remark

\( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
Performs processing in response to the SCI3n reception interrupt.

**[Syntax]**

```c
void r_<Config_SCION>_interrupt_receive(void);
```

*Remark*  
$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_SCI3n>_interrupt_error**

Performs processing in response to the SCI3\(n\) error interrupt.

### Syntax

```c
void r_<Config_SCI3n>_interrupt_error(void);
```

Remark \(n\) is the channel number.

### Argument(s)

None.

### Return value

None.
r_<Config_SCI3n>_interrupt_sendend

Performs processing in response to the SCI3n send end interrupt.

[Syntax]

```c
void r_<Config_SCI3n>_interrupt_sendend(void);
```

Remark  
\( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
**r_<Config_SCI3n>_callback_sendend**

Performs processing in response to the SCI3n communication interrupt.

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

\[
\text{r\_<Config\_SCI3n\>_\_interrupt\_send}
\]
corresponding to the SCI3 communication interrupt.

**[Syntax]**

\[
\text{void } \text{r\_<Config\_SCI3n\>_\_callback\_sendend(\text{void});}
\]

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_<Config_SCI3n>_callback_receiveend

Performs processing in response to the SCI3n reception interrupt.

Remark This API function is called as the callback routine of interrupt process
r_<Config_SCI3n>_interrupt_receive corresponding to the SCI3 error interrupt.

[Syntax]

```c
void r_<Config_SCI3n>_callback_receiveend(void);
```

Remark $n$ is the channel number.

[Argument(s)]

None.

[Return value]

None.
**r_<Config_SCI3n>_callback_error**

Performs processing in response to the SCI3\textit{n} error interrupt.

**Remark** This API function is called as the callback routine of interrupt process \( r_<\text{Config\_SCI3n\_interrupt\_error} \) corresponding to the SCI3 error interrupt.

**[Syntax]**

```c
void r_<Config_SCI3n>_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 err_type</td>
<td>Trigger for SCI3 error interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0xxxxxxxB: Transmit Data Register Empty</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x1xxxxxxB: Receive Data Register Full</td>
</tr>
<tr>
<td></td>
<td></td>
<td>xx1xxxxxB: Overrun Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>xxx1xxxxxB: Framing Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>xxxx1xxxxB: Parity Error</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
Usage example

This is an example for SCI30 to send and receive data in SCI3 asynchronous mode:
(Blue code is user code.)

r_cg_main.c

```c
/* Start user code for global. Do not edit comment generated here */
uint8_t tx_buf[6] = {0xAA, 0x55, 0x00, 0xFF, 0xCC, 0x33};
uint8_t rx_buf[6] = {0x88, 0x0, 0x00, 0x00, 0x00, 0x88};
/* End user code. Do not edit comment generated here */

void main(void)
{
    r_main_userinit();
    /* Start user code for main. Do not edit comment generated here */
    R_Config_SCI30_Start();
    R_Config_SCI30_Send(tx_buf, 6);
    while(transmitend_flag != 1);
    transmitend_flag = 0;

    R_Config_SCI30_Receive(rx_buf, 6);
    while(receiveendend_flag != 1);
    receiveendend_flag = 0;
    while(1);
    /* End user code. Do not edit comment generated here */
}
```

Config_SCI30_user.c

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

void r_Config_SCI30_callback_sendend(void)
{
    /* Start user code for r_Config_SCI30_callback_sendend. Do not edit comment generated here */
    transmitend_flag = 1;
    /* End user code. Do not edit comment generated here */
}

void r_Config_SCI30_callback_receiveend(void)
{
    /* Start user code for r_Config_SCI30_callback_receiveend. Do not edit comment generated here */
    receiveendend_flag = 1;
    /* End user code. Do not edit comment generated here */
}
```
### 3.2.36 MSPI Master

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

#### Table 3.36 API Functions: [MSPI Master]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_.&lt;Config_MSPInm&gt;_Create</td>
<td>Performs initialization necessary to control the MSPI Master functions</td>
</tr>
<tr>
<td>R_.&lt;Config_MSPInm&gt;_Start</td>
<td>Enables MSPIn</td>
</tr>
<tr>
<td>R_.&lt;Config_MSPInm&gt;_Stop</td>
<td>Disables MSPIn</td>
</tr>
<tr>
<td>R_.&lt;Config_MSPInm&gt;_Send</td>
<td>Starts MSPIn data transmission</td>
</tr>
<tr>
<td>R_.&lt;Config_MSPInm&gt;_Receive</td>
<td>Starts MSPIn data reception</td>
</tr>
<tr>
<td>R_.&lt;Config_MSPInm&gt;_Software_Trigger</td>
<td>Sets channel enable and start trigger</td>
</tr>
<tr>
<td>R_.&lt;Config_MSPInm&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the MSPI Master functions</td>
</tr>
<tr>
<td>r_.&lt;Config_MSPInm&gt;_channelnm_interrupt_send</td>
<td>Channel send interrupt</td>
</tr>
<tr>
<td>r_.&lt;Config_MSPInm&gt;_channelnm_interrupt_receive</td>
<td>Channel receive interrupt</td>
</tr>
<tr>
<td>r_.&lt;Config_MSPInm&gt;_Callback_Interrupt_Send</td>
<td>MSPIn channelm responses communication interrupt</td>
</tr>
<tr>
<td>r_.&lt;Config_MSPInm&gt;_Callback_Interrupt_Receive</td>
<td>MSPIn channelm responses reception interrupt</td>
</tr>
<tr>
<td>r_.&lt;Config_MSPInm&gt;_Callback_Interrupt_error</td>
<td>MSPIn channelm responses error interrupt</td>
</tr>
<tr>
<td>r_.&lt;Config_MSPInm&gt;_Callback_Interrupt_Frameend</td>
<td>Performs processing in response to the MSPInm frameend interrupt</td>
</tr>
<tr>
<td>r_.&lt;Config_MSPInm&gt;_callback_sendend</td>
<td>Performs processing in response to the MSPInm communication interrupt</td>
</tr>
<tr>
<td>r_.&lt;Config_MSPInm&gt;_callback_receiveend</td>
<td>Performs processing in response to the MSPInm reception interrupt</td>
</tr>
<tr>
<td>r_.&lt;Config_MSPInm&gt;_callback_error</td>
<td>Performs processing in response to the MSPInm error interrupt</td>
</tr>
</tbody>
</table>
**R_<Config_MSKInm>_Create**

Performs initialization necessary to control the MSPI Master functions.

**Remark**  This API function is called by **R_Systeminit** function.

**[Syntax]**

```c
void R_<Config_MSKInm>_Create(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_MSPInm>_Start

Enables SPI\(n\).

[Syntax]

```c
void R_<Config_MSPInm>_Start();
```

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

[Argument(s)]

None.

[Return value]

None.
### R\_<Config_MSPI\_nm>\_Stop

Disables MSPI\_n.

#### Syntax

```c
void R_<Config_MSPInm>_Stop(void);
```

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

#### Argument(s)

None.

#### Return value

None.
### R_<Config_MSKPlnm>_Send

Starts MSPI\(m\) data transmission.

#### Remark

**Remark 1**  This API function repeats the MSPI\(n\) transmission from the buffer specified in argument tx\_buf the number of times specified in frame count.

**Remark 2**  R_<Config_MSKPlnm>_Start must be called before this API function is called.

#### [Syntax]

<table>
<thead>
<tr>
<th>MD_STATUS</th>
<th>R_&lt;Config_MSKPlnm&gt;_Send (const uint8_t * tx_buf);</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_STATUS</td>
<td>R_&lt;Config_MSKPlnm&gt;_Send (const uint16_t * tx_buf);</td>
</tr>
<tr>
<td>MD_STATUS</td>
<td>R_&lt;Config_MSKPlnm&gt;_Send (const uint32_t * tx_buf);</td>
</tr>
</tbody>
</table>

#### [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 * tx_buf</td>
<td>Pointer to a buffer storing the transmission data (1 byte-level)</td>
</tr>
<tr>
<td>I</td>
<td>const uint16_t * tx_buf</td>
<td>Pointer to a buffer storing the transmission data (2 byte-level)</td>
</tr>
<tr>
<td>I</td>
<td>const uint32_t * tx_buf</td>
<td>Pointer to a buffer storing the transmission data (4 byte-level)</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_{MSPI_{nm}}} Receive

Starts MSPI<sub>nm</sub> data reception.

**Remark 1.** This API function performs MSPI<sub>n</sub> 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 \(R_{<Config_{MSPI_{nm}}}_{Start}\) is then called.

#### [Syntax]

```c
MD_STATUS R_{Config_{MSPI_{nm}}}_{Receive}(uint8_t * rx_{buf});
MD_STATUS R_{Config_{MSPI_{nm}}}_{Receive}(uint16_t * rx_{buf});
MD_STATUS R_{Config_{MSPI_{nm}}}_{Receive}(uint32_t * rx_{buf});
```

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>uint8_t * (rx_{buf})</td>
<td>Pointer to a buffer storing the transmission data(1 byte-level)</td>
</tr>
<tr>
<td>O</td>
<td>uint16_t * (rx_{buf})</td>
<td>Pointer to a buffer storing the transmission data(2 byte-level)</td>
</tr>
<tr>
<td>O</td>
<td>Uint32_t * (rx_{buf})</td>
<td>Pointer to a buffer storing the transmission data(4 byte-level)</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_\text{<Config\_MSPI\textit{nm}\_Software\_Trigger}}

Sets channel enable and start trigger.

[Syntax]

\begin{verbatim}
void R_\text{<Config\_MSPI\textit{nm}\_Software\_Trigger}(void);
\end{verbatim}

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

[Argument(s)]

None.

[Return value]

None.
**R_<Config_MSPInm>_Create_UserInit**

Performs user-defined initialization relating to the MSPIn functions.

**Remark**  This API function is called as the R_<Config_MSPInm>_Create callback routine.

**[Syntax]**

```c
void R_<Config_MSPInm>_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_&lt;Config_MSPI&gt;&lt;nm&gt;_channel&lt;m&gt;中断发送</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>功能</strong></td>
</tr>
</tbody>
</table>

**[Syntax]**

```c
void r_<Config_MSPI><nm>_channel<m>中断发送(void);
```

**Remark**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_\langle Config\_MSPI\text{nm}\rangle\_channel\text{nm}\_interrupt\_receive**

MSP\text{in} channel \text{m} receive interrupt.

**[Syntax]**

```c
void r_\langle Config\_MSPI\text{nm}\rangle\_channel\text{nm}\_interrupt\_receive(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_MSPI_{nm}>_Callback_Interrupt_Send**

Performs processing in response to the MSPI_{nm} communication interrupt.

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

**[Syntax]**

```c
void r_<Config_MSPI_{nm}>_Callback_Interrupt_Send(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_MSPI{nm}>_Callback_Interrupt_Receive**

Performs processing in response to the MSP{nm} reception interrupt.

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

**[Syntax]**

```c
void r_<Config_MSPI{nm}>_Callback_Interrupt_Receive(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_MSKInm>_Callback_Interrupt_Error**

Performs processing in response to the MSPI error interrupt.

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

**[Syntax]**

```c
void r_<Config_MSKInm>_Callback_Interrupt_Error(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
### r_<Config_MSPInm>Callback_Interrupt_Frameend

Performs processing in response to the MSPI<sub>n</sub>m frameend interrupt.

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

#### [Syntax]

```c
void r_<Config_MSPInm>_callback_frameend(void);
```

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

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

#### [Return value]
None.
**r_<Config_MSPI\(nm\)>_callback_sendend**

Performs processing in response to the MSPI\(nm\) communication interrupt.

**Remark**  
This API function is called as the callback routine of interrupt process \(\text{r_<Config_MSPI\(nm\)>_Callback\_Interrupt\_Send}\) corresponding to the MSPI\(nm\) communication interrupt.

**[Syntax]**

```c
void r_<Config_MSPI\(nm\)>_callback_sendend(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_<Config_MSPI<nm>>_callback_receiveend

Performs processing in response to the MSPI<nm> reception interrupt.

Remark This API function is called as the callback routine of interrupt process
r_<Config_MSPI<nm>>_Callback_Interrupt_Receive corresponding to the MSPI<nm> error
interrupt.

[Syntax]

void r_<Config_MSPI<nm>>_callback_receiveend(void);

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

[Argument(s)]
None.

[Return value]
None.
**r_<Config_MSPlnm>_callback_error**

Performs processing in response to the MSPlnm error interrupt.

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

\[ r_{<\text{Config\_MSPlnm}>\_Callback\_Interrupt\_Error} \] corresponding to the MSPlm error interrupt.

**[Syntax]**

```c
void r_<Config_MSPlnm>_callback_error(uint32_t err_type);
```

**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>I</td>
<td>uint32_t err_type</td>
<td>Trigger for MSPlm error interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>000100xxB: Transmission data consistency error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>000x001xB: Reception data CRC error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>000x00x1B: Reception data parity error</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
Usage example

This is an example for MSPI00 as master to send data and MSPI1 as slave to receive data:
(Blue code is user code.)

r_cg_main.c

/* Start user code for global. Do not edit comment generated here */
uint8_t transmitend_flag = 0;
uint8_t receiveend_flag = 0;

uint32_t tx_buf[4] = {0x12345678, 0x9abcdef0, 0xfedcba98, 0x76543210};
uint32_t rx_buf[4] = {0};
/* End user code. Do not edit comment generated here */

void main(void)
{
    r_main_userinit();
    /* Start user code for main. Do not edit comment generated here */
    R_Config_MSPI00_Start();
    R_MSPI_Start_Interrupt_MSPI0TX();
    R_MSPI_Start_Interrupt_MSPI0FE();
    R_MSPI_Start_Interrupt_MSPI0ERR();
    R_Config_MSPI01_Start();

    R_Config_MSPI1_Receive(rx_buf);
    R_Config_MSPI00_Send(tx_buf);

    R_Config_MSPI1_Software_Trigger();
    R_Config_MSPI00_Software_Trigger();
    while(1);
    /* End user code. Do not edit comment generated here */
}
### 3.2.37 MSPI Slave

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

**Table 3.37 API Functions: [MSPI Slave]**

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;/Config_MSPI&gt;n&gt;_Create</td>
<td>Performs initialization necessary to control the MSPI Slave functions</td>
</tr>
<tr>
<td>R_&lt;/Config_MSPI&gt;n&gt;_Start</td>
<td>Enables MSPI&lt;n&gt;</td>
</tr>
<tr>
<td>R_&lt;/Config_MSPI&gt;n&gt;_Stop</td>
<td>Disables MSPI&lt;n&gt;</td>
</tr>
<tr>
<td>R_&lt;/Config_MSPI&gt;n&gt;_Send</td>
<td>Starts MSPI&lt;n&gt; data transmission</td>
</tr>
<tr>
<td>R_&lt;/Config_MSPI&gt;n&gt;_Receive</td>
<td>Starts MSPI&lt;n&gt; data reception</td>
</tr>
<tr>
<td>R_&lt;/Config_MSPI&gt;n&gt;_Software_Trigger</td>
<td>Set channel enable and start trigger</td>
</tr>
<tr>
<td>R_&lt;/Config_MSPI&gt;n&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the MSPI Slave functions</td>
</tr>
<tr>
<td>r_&lt;/Config_MSPI&gt;n&gt;_channeln0_interrupt_send</td>
<td>Performs processing in response to the MSPI&lt;n&gt; channel 0 send interrupt</td>
</tr>
<tr>
<td>r_&lt;/Config_MSPI&gt;n&gt;_channeln0_interrupt_receive</td>
<td>Performs processing in response to the MSPI&lt;n&gt; channel 0 receive interrupt</td>
</tr>
<tr>
<td>r_&lt;/Config_MSPI&gt;n&gt;_interrupt_send</td>
<td>Performs processing in response to the MSPI Slave communication interrupt</td>
</tr>
<tr>
<td>r_&lt;/Config_MSPI&gt;n&gt;_interrupt_receive</td>
<td>Performs processing in response to the MSPI&lt;n&gt; reception interrupt</td>
</tr>
<tr>
<td>r_&lt;/Config_MSPI&gt;n&gt;_interrupt_error</td>
<td>Performs processing in response to the MSPI&lt;n&gt; error interrupt</td>
</tr>
<tr>
<td>r_&lt;/Config_MSPI&gt;n&gt;_interrupt_frameend</td>
<td>Performs processing in response to the MSPI&lt;n&gt; frame end interrupt</td>
</tr>
<tr>
<td>r_&lt;/Config_MSPI&gt;n&gt;_callback_sendend</td>
<td>Performs processing in response to the MSPI&lt;n&gt; communication interrupt</td>
</tr>
<tr>
<td>r_&lt;/Config_MSPI&gt;n&gt;_callback_receiveend</td>
<td>Performs processing in response to the MSPI&lt;n&gt; reception interrupt</td>
</tr>
<tr>
<td>r_&lt;/Config_MSPI&gt;n&gt;_callback_error</td>
<td>Performs processing in response to the MSPI&lt;n&gt; error interrupt</td>
</tr>
</tbody>
</table>
R_<Config_MSPIn>_Create

Performs initialization necessary to control the MSPI Slave functions.

Remark This API function is called by R_Systeminit function.

[Syntax]

```c
void R_<Config_MSPIn>_Create(void);
```

Remark \( n \) is the unit number.

[Argument(s)]

None.

[Return value]

None.
Enables MSPln.

**[Syntax]**

```c
void R_<Config_MSPln>_Start(void);
```

**Remark**

\( n \) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
Disables MSPln.

[Syntax]

```c
void R_<Config_MSBIn>_Stop(void);
```

Remark  

*n* is the unit number.

[Argument(s)]

None.

[Return value]

None.
**R_<Config_MSPIn>_Send**

Starts MSPI\(n\) data transmission.

**Remark1**  
This API function repeats the MSPI\(n\) transmission from the buffer specified in argument \(tx\_buf\) the number of times specified in frame count.

**Remark2**  
\(R_{<Config\_MSPIn>_Start}\) must be called before this API function is called.

**[Syntax]**

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_STATUS R_&lt;Config_MSPI(n)&gt;_Send (const uint8_t * tx_buf);</td>
</tr>
<tr>
<td>MD_STATUS R_&lt;Config_MSPI(n)&gt;_Send (const uint16_t * tx_buf);</td>
</tr>
<tr>
<td>MD_STATUS R_&lt;Config_MSPI(n)&gt;_Send (const uint32_t * tx_buf);</td>
</tr>
</tbody>
</table>

Remark  
\(n\) is the unit 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 * tx_buf</td>
<td>Pointer to a buffer storing the transmission data (1 byte-level)</td>
</tr>
<tr>
<td>I</td>
<td>const uint16_t * tx_buf</td>
<td>Pointer to a buffer storing the transmission data (2 byte-level)</td>
</tr>
<tr>
<td>I</td>
<td>const uint32_t * tx_buf</td>
<td>Pointer to a buffer storing the transmission data (4 byte-level)</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_MSPI_n>_Receive

Starts MSPI<sub>n</sub> data reception.

Remark1  This API function performs MSPI<sub>n</sub> reception the number of times specified by the frame count, and stores the data in the buffer specified by the argument `rx_buf`.

Remark2  Starts after this API function is called, and `R_<Config_MSPI_n>_Start` is then called.

**[Syntax]**

```
MD_STATUS R_<Config_MSPI_n>_Receive (uint8_t * rx_buf);
MD_STATUS R_<Config_MSPI_n>_Receive (uint16_t * rx_buf);
MD_STATUS R_<Config_MSPI_n>_Receive (uint32_t * rx_buf);
```

Remark  `n` is the unit 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 * <code>rx_buf</code></td>
<td>Pointer to a buffer storing the transmission data (1 byte-level)</td>
</tr>
<tr>
<td>O</td>
<td>uint16_t * <code>rx_buf</code></td>
<td>Pointer to a buffer storing the transmission data (2 byte-level)</td>
</tr>
<tr>
<td>O</td>
<td>Uint32_t * <code>rx_buf</code></td>
<td>Pointer to a buffer storing the transmission data (4 byte-level)</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_MSPI\textsubscript{n}>_Software_Trigger

Sets channel enable and start trigger.

**Syntax**

```c
void R_<Config_MSPI\textsubscript{n}>_Software_Trigger(void);
```

Remark \( n \) is the unit number.

**Argument(s)**

None.

**Return value**

None.
**R_{<\text{Config}_\text{MSPI}n>}_{\text{Create}_\text{UserInit}}**

Performs user-defined initialization relating to the MSPI\text{n} functions.

**Remark**  This API function is called as the \text{R}_{<\text{Config}_\text{MSPI}n>}_{\text{Create}} callback routine.

**[Syntax]**

```c
void _R_{<\text{Config}_\text{MSPI}n>}_{\text{Create}_\text{UserInit}}(void);
```

**Remark**  \text{n} is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
### r_<Config_MSPI>n>_channeln0_interrupt_send

Performs processing in response to the MSPI channel 0 send interrupt.

**[Syntax]**

```c
void r_<Config_MSPI>n>_channeln0_interrupt_send(void);
```

**Remark**

\( n \) is 0, 1.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_MSPIn>_channeln0_interrupt_receive**

Performs processing in response to the MSPI\(n\) channel 0 receive interrupt.

**[Syntax]**

```c
void r_<Config_MSPIn>_channeln0_interrupt_receive(void);
```

Remark  
\(n\) is 0, 1.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_{<Config_MSPI}\text{n}>\_interrupt\_send**

Performs processing in response to the MSPI\text{n} communication interrupt.

**[Syntax]**

```c
void r_{<Config_MSPI}\text{n}>\_interrupt\_send(void);
```

Remark \( n \) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_MSPIn>_interrupt_receive**

Performs processing in response to the MSPI

**[Syntax]**

```c
void r_<Config_MSPIn>_interrupt_receive(void);
```

Remark  

`n` is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<<Config_MSPIn>>_interrupt_error**

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

**[Syntax]**

```c
void r_<<Config_MSPIn>>_interrupt_error(void);
```

Remark \( n \) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_MSPIn>_interrupt_frameend**

Performs processing in response to the MSPI\textsubscript{n} frame end interrupt.

**[Syntax]**

```c
void r_<Config_MSPIn>_interrupt_frameend(void);
```

Remark \( n \) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_<Config_MSPI>n>_callback_sendend

Performs processing in response to the MSPI\textsubscript{n} communication interrupt.

Remark This API function is called as the callback routine of interrupt process \texttt{r_<Config_MSPI>n>_interrupt_send} corresponding to the MSPI\textsubscript{n} communication interrupt.

**[Syntax]**

```c
void r_<Config_MSPI>n>_callback_sendend(void);
```

Remark \( n \) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_{MSPIn}>_callback_receiveend**

Performs processing in response to the MSPIn reception interrupt.

**Remark**  
This API function is called as the callback routine of interrupt process  
`r_<Config_{MSPIn}>_interrupt_receive` corresponding to the MSPIn error interrupt.

**[Syntax]**

```c
void r_<Config_{MSPIn}>_callback_receiveend(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
r_<Config_MSPI_n>_callback_error

Performs processing in response to the MSPI_n error interrupt.

Remark This API function is called as the callback routine of interrupt process r_<Config_MSPI_n>_interrupt_error corresponding to the MSPI_n error interrupt.

[Syntax]

```c
void r_<Config_MSPI_n>_callback_error(uint32_t err_type);
```

Remark n is the unit 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 MSPI_n error interrupt  
 000100xxB: Transmission data consistency error  
 000x001xB: Reception data CRC error  
 000x00x1B: Reception data parity error |

[Return value]

None.
Usage example

Refer to MSPI master Usage example.
### 3.2.38 Real-Time Clock

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

#### Table 3.38 API Functions: [Real-Time Clock]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R.&lt;Config_RTCAn&gt;_Create</td>
<td>Performs initialization necessary to control the real-time clock</td>
</tr>
<tr>
<td>R.&lt;Config_RTCAn&gt;_Start</td>
<td>Starts the count of the real-time clock (year, month, weekday, day, hour, minute, second)</td>
</tr>
<tr>
<td>R.&lt;Config_RTCAn&gt;_Stop</td>
<td>Ends the count of the real-time clock (year, month, weekday, day, hour, minute, second)</td>
</tr>
<tr>
<td>R.&lt;Config_RTCAn&gt;_Set_HourSystem</td>
<td>Sets the clock type (12-hour or 24-hour clock) of the real-time clock</td>
</tr>
<tr>
<td>R.&lt;Config_RTCAn&gt;_Get_CounterBufferValue</td>
<td>Reads counter buffer register</td>
</tr>
<tr>
<td>R.&lt;Config_RTCAn&gt;_Get_CounterDirectValue</td>
<td>Reads the counter register directly</td>
</tr>
<tr>
<td>R.&lt;Config_RTCAn&gt;_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.&lt;Config_RTCAn&gt;_Get_SubCounterValue</td>
<td>Reads the buffer register of the sub-counter</td>
</tr>
<tr>
<td>R.&lt;Config_RTCAn&gt;_Set_ErrorCorrectionValue</td>
<td>Sets the error correction value</td>
</tr>
<tr>
<td>R.&lt;Config_RTCAn&gt;_Set_SubCounterCompareValue</td>
<td>Sets the comparison value of the sub-counter</td>
</tr>
<tr>
<td>R.&lt;Config_RTCAn&gt;_Get_AlarmValue</td>
<td>Reads the alarm conditions (weekday, hour, minute)</td>
</tr>
<tr>
<td>R.&lt;Config_RTCAn&gt;_Set_AlarmValue</td>
<td>Sets the alarm conditions (weekday, hour, minute)</td>
</tr>
<tr>
<td>R.&lt;Config_RTCAn&gt;_Set_AlarmOn</td>
<td>Starts the alarm interrupt function</td>
</tr>
<tr>
<td>R.&lt;Config_RTCAn&gt;_Set_AlarmOff</td>
<td>Ends the alarm interrupt function</td>
</tr>
<tr>
<td>R.&lt;Config_RTCAn&gt;_Set_ConstPeriodInterruptOn</td>
<td>Sets the cycle of the periodic interrupts, then starts the periodic interrupt function</td>
</tr>
<tr>
<td>R.&lt;Config_RTCAn&gt;_Set_ConstPeriodInterruptOff</td>
<td>Ends the periodic interrupt function</td>
</tr>
<tr>
<td>R.&lt;Config_RTCAn&gt;_Set_1secondInterruptOn</td>
<td>Starts the 1 second interrupt function</td>
</tr>
<tr>
<td>R.&lt;Config_RTCAn&gt;_Set_1secondInterruptOff</td>
<td>Ends the 1 second interrupt function</td>
</tr>
<tr>
<td>R.&lt;Config_RTCAn&gt;_Set_RTCA1HZOn</td>
<td>Enables output of the correction clock (1 Hz) to the RTC1HZ pin</td>
</tr>
<tr>
<td>R.&lt;Config_RTCAn&gt;_Set_RTCA1HZOff</td>
<td>Enables output of the correction clock (1 Hz) to the RTC1HZ pin</td>
</tr>
<tr>
<td>R.&lt;Config_RTCAn&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the real-time clock</td>
</tr>
<tr>
<td>r.&lt;Config_RTCAn&gt;_interrupt_alarm</td>
<td>Performs processing in response to the alarm interrupt</td>
</tr>
<tr>
<td>r.&lt;Config_RTCAn&gt;_interrupt_periodic</td>
<td>Performs processing in response to the periodic interrupt</td>
</tr>
<tr>
<td>r.&lt;Config_RTCAn&gt;_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 by R_Systeminit function.

[Syntax]

```c
void R_{Config_RTCAn}_Create(void);
```

Remark   \( n \) is the unit number.

[Argument(s)]

None.

[Return value]

None.
**R_<Config_RTCAn>_Start**

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

**[Syntax]**

```c
void R_<Config_RTCAn>_Start(void);
```

Remark  
$n$ is the unit number.

**[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);
```

Remark  

\( n \) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_\text{<Config_RTCn>\_Set\_HourSystem}**

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

**[Syntax]**

```c
MD_STATUS R_\text{<Config_RTCn>\_Set\_HourSystem}(rtc_hour_system_t \text{hour\_system});
```

Remark \(n\) is the unit 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>rtc_hour_system_t \text{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

 Reads counter buffer register.

#### Syntax

```c
MD_STATUS R\_<Config_RTCAn>_Get_CounterBufferValue(rtc_counter_value_t * const counter_read_val);
```

Remark  
\( n \) is the unit 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>rtc_counter_value_t * const</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

Reads the counter register directly.

**[Syntax]**

\[
\text{MD\_STATUS} \quad \text{R}_{<}\text{Config\_RTCAn}>\_\text{Get\_CounterDirectValue}(\text{rtc\_counter\_value\_t} \ast \text{const}, \text{counter\_read\_val});
\]

Remark \( n \) is the unit 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>\text{rtc_counter_value_t} \ast \text{const} \text{counter_read_val}</td>
<td>Pointer to the structure storing the read count value</td>
</tr>
</tbody>
</table>

Remark For structure \text{rtc\_counter\_value\_t}, see \text{R}_{<}\text{Config\_RTCAn}>\_\text{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]**

```
MD_STATUS R_<Config_RTCAn>_Set.CounterValue(rtc_counter_value_t counter_write_val);
```

**Remark**  
$n$ is the unit 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>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**

Reads the buffer register of the sub-counter.

**Syntax**

```
MD_STATUS R_<Config_RTCAn>_Get_SubCounterValue(uint32_t * const subcounter_read_val);
```

Remark  
$n$ is the unit 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 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\_RTCAn>\_Set\_ErrorCorrectionValue

Set the error correction value.

#### [Syntax]

```
MD_STATUS R\_\_<Config\_RTCAn>\_Set\_ErrorCorrectionValue(uint8_t * const errorcorrection_write_val);
```

Remark  
\( n \) is the unit 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 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>

Set the comparison value of the sub-counter.

**[Syntax]**

```
MD_STATUS R_<Config_RTCAn>_Set_SubCounterCompareValue(uint32_t * const subcompare_write_val);
```

Remark

\( n \) is the unit 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 * const subcompare_write_val</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]

void R_<Config_RTCAn>_Get_AlarmValue(rtc_alarm_value_t * const alarm_val);

Remark  \( n \) is the unit 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>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_AlmValue**

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

**[Syntax]**

```c
void R_<Config_RTCAn>_Set_AlmValue(rtc_alarm_value_t alarm_val);
```

Remark $n$ is the unit 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>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_AlmValue.

**[Return value]**

None.
R_{<Config_RTCAn>_Set_AlarmOn}

Starts the alarm interrupt function.

[Syntax]

```c
void R_{<Config_RTCAn>_Set_AlarmOn}(void);
```

Remark  
\(n\) is the unit number.

[Argument(s)]

None.

[Return value]

None.
### R\_<Config_RTC\text{n}>\_Set_AlarmOff

Ends the alarm interrupt function.

**[Syntax]**

```c
void R\_<Config_RTC\text{n}>\_Set_AlarmOff(void);
```

**Remark**

\textit{n} is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_RTCAn>_Set_ConstPeriodInterruptOn**

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

**[Syntax]**

```
MD_STATUS R_<Config_RTCAn>_Set_ConstPeriodInterruptOn(rtc_int_period_t period);
```

Remark  
\( n \) is the unit 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>rtc_int_period_t period</td>
<td>Interrupt INTRTC cycle</td>
</tr>
<tr>
<td></td>
<td>QUARTERSEC</td>
<td>0.25 seconds</td>
</tr>
<tr>
<td></td>
<td>HALFSEC</td>
<td>0.5 seconds</td>
</tr>
<tr>
<td></td>
<td>ONESEC</td>
<td>1 second</td>
</tr>
<tr>
<td></td>
<td>ONEMIN</td>
<td>1 minute</td>
</tr>
<tr>
<td></td>
<td>ONEHOUR</td>
<td>1 hour</td>
</tr>
<tr>
<td></td>
<td>ONEDAY</td>
<td>1 day</td>
</tr>
<tr>
<td></td>
<td>ONEMONTH</td>
<td>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_CstPeriodInterruptOff

Ends the periodic interrupt function.

**[Syntax]**

```c
void R_<Config_RTCAn>_Set_CstPeriodInterruptOff(void);
```

Remark  $n$ is the unit number.

**[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);
```

Remark  

$n$ is the unit number.

[Argument(s)]

None.

[Return value]

None.
**R\_<Config_RTCAn>_Set_1secondInterruptOff**

Ends the 1 second interrupt function.

**[Syntax]**

```c
void R_<Config_RTCAn>_Set_1secondInterruptOff(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

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

**Syntax**

```c
void R_<Config_RTCAn>_Set_RTCA1HZOn(void);
```

*Remark*  
$n$ is the unit number.

**Argument(s)**  
None.

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

**Syntax**

```
void R_<_Config_RTCAn>_Set_RTCA1HZOff(void);
```

Remark  
\( n \) is the unit number.

**Argument(s)**

None.

**Return value**

None.
**R_{<Config_RTCAn>_Create_UserInit}**

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

**Remark**  This API function is called as the R_{<Config_RTCAn>_Create} callback routine.

**[Syntax]**

```c
void R_{<Config_RTCAn>_Create_UserInit}(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
Performs processing in response to the alarm interrupt.

**Syntax**

```c
void r_<Config_RTCAn>_interrupt_alarm(void);
```

Remark: `n` is the unit number.

**Argument(s)**

None.

**Return value**

None.
**r.<Config_RTC\textsubscript{\textit{n}}>_interrupt_periodic**

Performs processing in response to the periodic interrupt.

**[Syntax]**

```c
void r.<Config_RTC\textsubscript{\textit{n}}>_interrupt_periodic(void);
```

Remark \( n \) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_<Config_RTCAn>_interrupt_1second

Performs processing in response to the 1 second interrupt.

[Syntax]

void r_<Config_RTCAn>_interrupt_1second(void);

Remark

n is the unit number.

[Argument(s)]

None.

[Return value]

None.
### 3.2.39 Error Control Module

Below is a list of API functions output by the Code Generator for error control module use.

#### Table 3.39 API Functions: [Error Control Module]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>R_&lt;Config_ECM&gt;_Create</code></td>
<td>Performs initialization necessary to control the error control module</td>
</tr>
<tr>
<td><code>R_&lt;Config_ECM&gt;_Start</code></td>
<td>Starts the error control module</td>
</tr>
<tr>
<td><code>R_&lt;Config_ECM&gt;_Stop</code></td>
<td>Stops the error control module</td>
</tr>
<tr>
<td><code>R_&lt;Config_ECM&gt;_Set_Master_Error</code></td>
<td>Sets master error</td>
</tr>
<tr>
<td><code>R_&lt;Config_ECM&gt;_Set_Checker_Error</code></td>
<td>Sets checker error</td>
</tr>
<tr>
<td><code>R_&lt;Config_ECM&gt;_Clear_Master_Error</code></td>
<td>Clear master error</td>
</tr>
<tr>
<td><code>R_&lt;Config_ECM&gt;_Clear_Checker_Error</code></td>
<td>Clear checker error</td>
</tr>
<tr>
<td><code>R_&lt;Config_ECM&gt;_Delay_Timer_Start</code></td>
<td>Starts the delay timer</td>
</tr>
<tr>
<td><code>R_&lt;Config_ECM&gt;_Delay_Timer_Stop</code></td>
<td>Stops the delay timer</td>
</tr>
<tr>
<td><code>R_&lt;Config_ECM&gt;_Pseud_XXXX_Start</code></td>
<td>Starts ECM pseudo debug operation for XXXX function</td>
</tr>
<tr>
<td><code>R_&lt;Config_ECM&gt;_Pseudo_XXXX_Stop</code></td>
<td>Stops ECM pseudo debug operation for XXXX function</td>
</tr>
<tr>
<td><code>R_&lt;Config_ECM&gt;_Create_UserInit</code></td>
<td>Performs user-defined initialization relating to the error control module functions</td>
</tr>
<tr>
<td><code>r_&lt;Config_ECM&gt;_ecmmi_interrupt_pek</code></td>
<td>Performs processing in response to the ECM maskable interrupt for PE k</td>
</tr>
<tr>
<td><code>r_&lt;Config_ECM&gt;_mi_interrupt_pek</code></td>
<td>Performs processing in response to the ECM maskable interrupt for PE k</td>
</tr>
<tr>
<td><code>r_&lt;Config_ECM&gt;_dclsmi_interrupt_pek</code></td>
<td>Performs processing in response to the DCLS error interrupt for PE k</td>
</tr>
</tbody>
</table>
**R_<Config_ECM>_Create**

Performs initialization necessary to control the error control module.

**Remark**  This API function is called by **R_Systeminit** function.

**[Syntax]**

```c
void R_<Config_ECM>_Create(void);
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_ECM>_Start

Starts the error control module.

[Syntax]

```c
void R_<Config_ECM>_Start(void);
```

[Argument(s)]

None.

[Return value]

None.
R_<Config ECM>_Stop

Stops the error control module.

[Syntax]

```c
void R_<Config ECM>_Stop(void);
```

[Argument(s)]
None.

[Return value]
None.
R_<Config_ECM>_Set_Master_Error

Sets the master error.

**[Syntax]**

```c
void R_<Config_ECM>_Set_Master_Error(void);
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_ECM>_Set_Checker_Error

Sets the checker error.

[Syntax]

void R_<Config_ECM>_Set_Checker_Error(void);

[Argument(s)]

None.

[Return value]

None.
R_<Config_ECM>_Clear_Master_Error

Clears the master error.

[Syntax]

```c
void R_<Config_ECM>_Clear_Master_Error(void);
```

[Argument(s)]

None.

[Return value]

None.
R_\textless Config\_ECM\textgreater\_Clear\_Checker\_Error

Clears the checker error.

**[Syntax]**

```c
void R_\textless Config\_ECM\textgreater\_Clear\_Checker\_Error(void);
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_ECM>_Delay_Timer_Start

Starts the delay timer.

[Syntax]

```c
void R_<Config_ECM>_Delay_Timer_Start(void);
```

[Argument(s)]

None.

[Return value]

None.
R_<Config_ECM>_Delay_Timer_Stop

Stops the delay timer.

[Syntax]

```c
void R_<Config_ECM>_Delay_Timer_Stop(void);
```

[Argument(s)]

None.

[Return value]

None.
R_<Config_ECM>_Pseudo_XXXX_Start

Starts ECM pseudo debug operation for XXXX function.

[Syntax]

```c
void R_<Config_ECM>_Pseudo_XXXX_Start(void);
```

Remark XXXX represent the module/function in Table 3.39.1 XXXX module/function name.

### Table 3.39.1 ECM pseudo error name

<table>
<thead>
<tr>
<th>XXXX name</th>
<th>XXXX name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delay_Timer_Overflow</td>
<td>ECM_Compare_Error</td>
</tr>
<tr>
<td>Activation_Production_Test_Mode</td>
<td>Activation_Normal Operation_Mode</td>
</tr>
<tr>
<td>Deactivation_Normal_Operation_Mode</td>
<td>Serial_Programming_Mode</td>
</tr>
<tr>
<td>Activation_User_Boot_Mode</td>
<td>Deactivation_User_Boot_Mode</td>
</tr>
<tr>
<td>Mode_Check_Error</td>
<td>Flash_Access_Error</td>
</tr>
<tr>
<td>FACI_Reset_Transfer_Error</td>
<td>BiST_Parameter_Transfer_Error</td>
</tr>
<tr>
<td>DTS_Compare_Error</td>
<td>BUS_Bridge_Compare_Error_SDMA</td>
</tr>
<tr>
<td>BUS_Bridge_Compare_Error</td>
<td>Element_Bus_Routing_Error</td>
</tr>
<tr>
<td>I_Bus_Routing_Error</td>
<td>P_Bus_Routing_Error</td>
</tr>
<tr>
<td>CRAM_Bus_Routing_Error</td>
<td>System_Bus_Routing_Error</td>
</tr>
<tr>
<td>Global_Flash_Bus_Routing_Error</td>
<td>Local_Flash_Bus_Routing_Error</td>
</tr>
<tr>
<td>Clock_Monitor_Error_MOSC</td>
<td>Clock_Monitor_Error_HSintOSC</td>
</tr>
<tr>
<td>Clock_Monitor_Error_LSiOSC</td>
<td>Clock_Monitor_Error_CLK_LSB</td>
</tr>
<tr>
<td>Clock_Monitor_Error_CLK_UHSB</td>
<td>Clock_Monitor_Error_CLK_HBUS</td>
</tr>
<tr>
<td>OSTMn_Interrupt_Error</td>
<td>ADCJn_Parity_Error</td>
</tr>
<tr>
<td>Temperature_Sensor_Error</td>
<td>Code_Flash_Address_Parity_Error</td>
</tr>
<tr>
<td>Code_Flash_ECC_Uncorrectable_Error</td>
<td>Code_Flash_ECC_Correctable_Error</td>
</tr>
<tr>
<td>Code_Flash_ECC_Overflow_Error</td>
<td>Data_Flash_ECC_Uncorrectable_Error</td>
</tr>
<tr>
<td>Data_Flash_ECC_Correctable_Error</td>
<td>Data_Flash_ECC_Overflow_Error</td>
</tr>
<tr>
<td>Local_RAM_ECC_Uncorrectable_Error</td>
<td>Local_RAM_ECC_Correctable_Error</td>
</tr>
<tr>
<td>Local_RAM_ECC_Overflow_Error</td>
<td>Cluster_RAM_ECC_Uncorrectable_Error</td>
</tr>
<tr>
<td>Cluster_RAM_ECC_Correctable_Error</td>
<td>Cluster_RAM_ECC_Overflow_Error</td>
</tr>
<tr>
<td>DTSRAM_ECC_Uncorrectable_Error</td>
<td>DTSRAM_ECC_Correctable_Error</td>
</tr>
<tr>
<td>DTSRAM_ECC_Overflow_Error</td>
<td>SDMACnRAM_ECC_Uncorrectable_Error</td>
</tr>
<tr>
<td>SDMACnRAM_ECC_Correctable_Error</td>
<td>FlexRayRAM_ECC_Uncorrectable_Error</td>
</tr>
<tr>
<td>FlexRayRAM_ECC_Correctable_Error</td>
<td>RS_CANFRAM_ECC_Uncorrectable_Error</td>
</tr>
<tr>
<td>RS_CANFRAM_ECC_Correctable_Error</td>
<td>MSPI_RAM_ECC_Uncorrectable_Error</td>
</tr>
<tr>
<td>MSPI_RAM_ECC_Correctable_Error</td>
<td>GTM_RAM_ECC_Uncorrectable_Error</td>
</tr>
<tr>
<td>GTM_RAM_ECC_Correctable_Error</td>
<td>Fast_Ethernet_RAM_ECC_Uncorrectable_Error</td>
</tr>
<tr>
<td>Fast_Ethernet_RAM_ECC_Correctable_Error</td>
<td>Gigabit_Ethernet_RAM_ECC_Uncorrectable_Error</td>
</tr>
<tr>
<td>Gigabit_Ethernet_RAM_ECC_Correctable_Error</td>
<td>MMCA_RAM_ECC_Uncorrectable_Error</td>
</tr>
<tr>
<td>MMCA_RAM_ECC_Correctable_Error</td>
<td>Peripheral_ECC_Overflow_Error</td>
</tr>
<tr>
<td>Data_Transfer_Path_EDC_Error</td>
<td>Data_Transfer_Path_ECC_Uncorrectable_Error</td>
</tr>
<tr>
<td>Data_Transfer_Path_ECC_Correctable_Error</td>
<td>CRAM_Guard_Error</td>
</tr>
</tbody>
</table>
### 3. API FUNCTIONS

<table>
<thead>
<tr>
<th>Argument(s)</th>
<th>None.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return value</td>
<td>None.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>69</th>
<th>I_Bus_Guard_Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>P_Bus_Guard_Error</td>
</tr>
<tr>
<td>71</td>
<td>H_Bus_Guard_Error</td>
</tr>
<tr>
<td>72</td>
<td>DTS_sDMAC_Transfer_Error</td>
</tr>
<tr>
<td>73</td>
<td>H_Bus_Master_Transfer_Error</td>
</tr>
<tr>
<td>74</td>
<td>External_Error_Input_n_Error</td>
</tr>
<tr>
<td>75</td>
<td>Software_Alarm_n_Error</td>
</tr>
<tr>
<td>76</td>
<td>DCLS_Compare_Error_PEk</td>
</tr>
<tr>
<td>77</td>
<td>Unintended_Debug_Enable_Detection_PEk</td>
</tr>
<tr>
<td>78</td>
<td>Watchdog_Timer_Error_PEk</td>
</tr>
<tr>
<td>79</td>
<td>Clock_Monitor_Error_PEk</td>
</tr>
<tr>
<td>80</td>
<td>PE_Guard_Error_PEk</td>
</tr>
<tr>
<td>81</td>
<td>Data_Access_Error_PEk</td>
</tr>
<tr>
<td>82</td>
<td>PE_Guard_Error_PEk_Access</td>
</tr>
<tr>
<td>83</td>
<td>Local_RAM_PEk_ECC_Uncorrectable_Error</td>
</tr>
<tr>
<td>84</td>
<td>Local_RAM_PEk_ECC_Correctable_Error</td>
</tr>
<tr>
<td>85</td>
<td>Local_RAM_PEk_ECC_Overflow_Error</td>
</tr>
<tr>
<td>86</td>
<td>Instruction_Cache_RAM_PEk_ECC_Uncorrectable_Error</td>
</tr>
</tbody>
</table>
**R_<Config_ECM>_Pseudo_XXXX_Stop**

Stops ECM pseudo debug operation for XXXX function.

**[Syntax]**

```c
void R_<Config_ECM>_Pseudo_XXXX_Stop(void);
```

**Remark**

XXXX represent the module/function in Table 3.39.1 XXXX module/function name.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_ECM>_Create_UserInit**

Performs user-defined initialization relating to the error control module functions.

Remark  This API functions is called as the **R_<Config_ECM>_Create** callback routine.

**[Syntax]**

```c
void R_<Config_ECM>_Create_UserInit(void);
```

**[Argument(s)]**

None.

**[Return value]**

None.
Performs processing in response to the ECM maskable interrupt for PE $k$.

**[Syntax]**

```c
void r_<Config_ECM>_ecmmi_interrupt_pek(void);
```

Remark $k$ is the CPU core number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_ECM>_mi_interrupt_pek**

Performs processing in response to the ECM maskable interrupt for PE \( k \).

**[Syntax]**

```c
void r_<Config_ECM>_mi_interrupt_pek(void);
```

Remark \( k \) is the CPU core number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_ECM>_dclsmi_interrupt_pek**

Performs processing in response to the DCLS error interrupt for PE k.

**[Syntax]**

```c
void r_<Config_ECM>_dclsmi_interrupt_pek(void);
```

Remark  
  k is the CPU core number.

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.40 GTM Clock

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

Table 3.40 API Functions: [GTM Clock]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_GTMClock&gt;_Create</td>
<td>Performs initialization necessary to control the GTM clock</td>
</tr>
<tr>
<td>R_GTMCCMn_Create</td>
<td>Performs initialization necessary to control the cluster</td>
</tr>
<tr>
<td>configuration module</td>
<td></td>
</tr>
<tr>
<td>R_&lt;Config_GTMClock&gt;_Create_Init</td>
<td>Performs user-defined initialization relating to the GTM clock</td>
</tr>
</tbody>
</table>
R_<Config_GTMClock>_Create

Performs initialization necessary to control the GTM clock.

Remark  This API function is called by R_Systeminit function.

[Syntax]

    void R_<Config_GTMClock>_Create(void);

[Argument(s)]

None.

[Return value]

None.
R_GTM_CCMn_Create

Performs initialization necessary to control the cluster configuration module.

Remark  This API function is called as the \texttt{R\_<Config\_GTM\_Clock>_Create} callback routine.

\textbf{[Syntax]}

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

Remark  \textit{n} is the CCM channel number.

\textbf{[Argument(s)]}

None.

\textbf{[Return value]}

None.
**R_<Config_GTM_Clock>_Create_UserInit**

Performs user-defined initialization relating to the GTM clock.

**Remark** This API function is called as the `R_<Config_GTM_Clock>_Create` callback routine.

**[Syntax]**

```c
void R_<Config_GTM_Clock>_Create_UserInit(void);
```

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.41 Time Base Unit

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

Table 3.41 API Functions: [Time Base Unit]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_TBU&gt;_Create</td>
<td>Performs initialization necessary to control the time base unit functions</td>
</tr>
<tr>
<td>R_&lt;Config_TBU&gt;_Start</td>
<td>Starts the count for time base unit</td>
</tr>
<tr>
<td>R_&lt;Config_TBU&gt;_Stop</td>
<td>Ends the count for time base unit</td>
</tr>
<tr>
<td>R_&lt;Config_TBU&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the time base unit functions</td>
</tr>
</tbody>
</table>
**R_Config_TBU_Create**

Performs initialization necessary to control the time base unit functions.

**Remark**  This API function is called by `R_Systeminit` function.

**[Syntax]**

```c
void R_Config_TBU_Create(void);
```

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TBU>_Start

Starts the count for time base unit

[Syntax]

void R_<Config_TBU>_Start(void);

[Argument(s)]

None.

[Return value]

None.
**R_<Config_TBU>_Stop**

Ends the count for time base unit

**[Syntax]**

```c
void R_<Config_TBU>_Stop(void);
```

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_TBU>_Create_UserInit**

Performs user-defined initialization relating to the time base unit functions.

**Remark**  This API function is called as the R_<Config_TBU>_Create callback routine.

**[Syntax]**

```c
void R_<Config_TBU>_Create_UserInit(void);
```

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.42 TIM Bit Compression Mode

Below is a list of API functions output by the Code Generator for TIM bit compression mode use.

Table 3.42 API Functions: [TIM Bit Compression Mode]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R.&lt;Config_TIMn&gt;_Create</td>
<td>Performs initialization necessary to control the TIM bit compression mode</td>
</tr>
<tr>
<td>R.&lt;Config_TIMn&gt;_Start</td>
<td>Starts compression processing for unit n</td>
</tr>
<tr>
<td>R.&lt;Config_TIMn&gt;_Stop</td>
<td>Stops compression processing for unit n</td>
</tr>
<tr>
<td>R.&lt;Config_TIMn&gt;_SetDetectedEdge</td>
<td>Sets detected edge number</td>
</tr>
<tr>
<td>R.&lt;Config_TIMn&gt;_Software_Reset</td>
<td>Stops channel operation and reset channel registers</td>
</tr>
<tr>
<td>R.&lt;Config_TIMn&gt;_NEWVAL_TriggerOn</td>
<td>Trigger NEWVAL bit in TIMn_CH0_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R.&lt;Config_TIMn&gt;_ECNTOFL_TriggerOn</td>
<td>Trigger ECNTOFL bit in TIMn_CH0_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R.&lt;Config_TIMn&gt;_CNTOFL_TriggerOn</td>
<td>Trigger CNTOFL bit in TIMn_CH0_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R.&lt;Config_TIMn&gt;_GPROFL_TriggerOn</td>
<td>Trigger GPROFL bit in TIMn_CH0_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R.&lt;Config_TIMn&gt;_TODETOFL_TriggerOn</td>
<td>Trigger TODETOFL bit in TIMn_CH0_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R.&lt;Config_TIMn&gt;_Channelm_GLITCHDETOFL_TriggerOn</td>
<td>Trigger GLITCHDET bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R.&lt;Config_TIMn&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the TIM bit compression mode</td>
</tr>
<tr>
<td>r.&lt;Config_TIMn&gt;_Callback_GTM_Error</td>
<td>Processing is performed according to the occurrence of a count error interrupt</td>
</tr>
<tr>
<td>r.&lt;Config_TIMn&gt;_share_interrupt</td>
<td>Performs processing in response to the TIMn interrupt</td>
</tr>
<tr>
<td>r.&lt;Config_TIMn&gt;_chm_share_interrupt</td>
<td>Performs processing in response to the TIMn channelm interrupt</td>
</tr>
</tbody>
</table>
R_\langle\text{Config\_TIMn}\rangle\_Create

Performs initialization necessary to control the TIM bit compression mode.

Remark This API function is called by \texttt{R\_Systeminit} function.

\begin{verbatim}
void R_\langle\text{Config\_TIMn}\rangle\_Create(void);

Remark \texttt{n} is the unit number.
\end{verbatim}

\[ \text{[Argument(s)]} \]
None.

\[ \text{[Return value]} \]
None.
R_\text{<Config\_TIMn>\_Start}

Starts compression processing for unit \(n\).

**[Syntax]**

\begin{verbatim}
void R_{\text{<Config\_TIMn>\_Start}}(void);
\end{verbatim}

Remark \(n\) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TIMn>_Stop

Stops compression processing for unit \( n \).

**[Syntax]**

```c
void R_<Config_TIMn>_Stop(void);
```

Remark \( n \) is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config.TIMn>_SetDetectedEdge**

Sets detected edge number.

**[Syntax]**

```c
void R_<Config.TIMn>_SetDetectedEdge (uint16_t number);
```

*Remark*  \( n \) is the unit 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 number</td>
<td>Detected edge number</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
R_<Config_TIMn>_Software_Reset

Stops channel operation and reset channel registers.

[Syntax]

```c
void R_<Config_TIMn>_Software_Reset(void);
```

Remark

\( n \) is the unit number.

[Argument(s)]

None.

[Return value]

None.
**R_<Config_TIMn>_NEWVAL_TriggerOn**

Trigger NEWVAL bit in TIMn_CH0_IRQ_NOTIFY register by software.

**Syntax**

```c
void R_<Config_TIMn>_NEWVAL_TriggerOn(void);
```

*Remark*  
*n* is the unit number.

**Argument(s)**

None.

**Return value**

None.
**R_<Config_TIMn>_ECNTOFL_TRIGGERON**

Trigger ECNTOFL bit in TIMn_CH0_IRQ_NOTIFY register by software.

**[Syntax]**

```
void R_<Config_TIMn>_ECNTOFL_TRIGGERON(void);
```

Remark

\[ n \] is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
### R_<Config_TIMn>_CNTOFL_TriggerOn

Trigger CNTOFL bit in TIMn_CH0_IRQ_NOTIFY register by software.

**[Syntax]**

```c
void R_<Config_TIMn>_CNTOFL_TriggerOn(void);
```

Remark  
$n$ is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TIMn>_GPROFL_TriggerOn

Trigger GPROFL bit in TIMn_CH0_IRQ_NOTIFY register by software.

[Syntax]

```c
void R_<Config_TIMn>_GPROFL_TriggerOn(void);
```

Remark

\(n\) is the unit number.

[Argument(s)]

None.

[Return value]

None.
**R_<Config_TIMn>_TODETOFL_TriggerOn**

Trigger TODETOFL bit in TIMn.CH0_IRQ_NOTIFY register by software.

**[Syntax]**

```c
void R_<Config_TIMn>_TODETOFL_TriggerOn(void);
```

Remark  
$n$ is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**R_{<Config_TIM\text{n}>}_Channelm,GLITCHDETOFL_TriggerOn**

Trigger GLITCHDETOFL bit in TIM\text{n}_CH\text{m}_IRQ_NOTIFY register by software

**[Syntax]**

```c
void R_{<Config_TIM\text{n}>}_Channelm,GLITCHDETOFL_TriggerOn(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TIMn>_Create_UserInit

Performs user-defined initialization relating to the TIM bit compression mode.

Remark This API function is called as the R_<Config_TIMn>_Create callback routine.

[Syntax]

```
void R_<Config_TIMn>_Create_UserInit(void);
```

Remark \( n \) is the unit number.

[Argument(s)]

None.

[Return value]

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

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

**[Syntax]**

```c
void r_<Config.TIMn>_Callback_GTM_Error(void);
```

Remark $n$ is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_TIMn>_share_interrupt**

Performs processing in response to the TIMn interrupt.

**[Syntax]**

```
void r_<Config_TIMn>_share_interrupt(void);
```

Remark

$n$ is the unit number.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_TIMn>_chm_share_interrupt**

Performs processing in response to the TIM\(n\) channel\(m\) interrupt.

**[Syntax]**

```c
void r_<Config_TIMn>_chm_share_interrupt(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.43 **TIM Gated Periodic Sampling Mode**

Below is a list of API functions output by the Code Generator for TIM gated periodic sampling mode use.

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_{&lt;Config_TIMn_m&gt;}_Create</td>
<td>Performs initialization necessary to control the TIM gated periodic sampling mode.</td>
</tr>
<tr>
<td>R_{&lt;Config_TIMn_m&gt;}_Start</td>
<td>Starts periodic sampling for channel $m$ in unit $n$</td>
</tr>
<tr>
<td>R_{&lt;Config_TIMn_m&gt;}_Stop</td>
<td>Stops periodic sampling for channel $m$ in unit $n$</td>
</tr>
<tr>
<td>R_{&lt;Config_TIMn_m&gt;}_UpdateElapsedClock</td>
<td>Updates elapsed clock number.</td>
</tr>
<tr>
<td>R_{&lt;Config_TIMn_m&gt;}_Software_Reset</td>
<td>Stops channel operation and reset channel registers</td>
</tr>
<tr>
<td>R_{&lt;Config_TIMn_m&gt;}_NEWVAL_TriggerOn</td>
<td>Trigger NEWVAL bit in TIM$n$ CH$m$ IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_{&lt;Config_TIMn_m&gt;}_ECNTOFL_TriggerOn</td>
<td>Trigger ECNTOFL bit in TIM$n$ CH$m$ IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_{&lt;Config_TIMn_m&gt;}_CNOFL_TriggerOn</td>
<td>Trigger CNOFL bit in TIM$n$ CH$m$ IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_{&lt;Config_TIMn_m&gt;}_GPROFL_TriggerOn</td>
<td>Trigger GPROFL bit in TIM$n$ CH$m$ IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_{&lt;Config_TIMn_m&gt;}_TODETOFL_TriggerOn</td>
<td>Trigger TODETOFL bit in TIM$n$ CH$m$ IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_{&lt;Config_TIMn_m&gt;}_GLITCHDETOFL_TriggerOn</td>
<td>Trigger GLITCHDET bit in TIM$n$ CH$m$ IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_{&lt;Config_TIMn_m&gt;}_Create_UserInit</td>
<td>Performs user-defined initialization relating to the TIM gated periodic sampling mode.</td>
</tr>
<tr>
<td>r_{&lt;Config_TIMn_m&gt;}_Callback_GTM_Error</td>
<td>Processing is performed according to the occurrence of a count error interrupt.</td>
</tr>
<tr>
<td>r_{&lt;Config_TIMn_m&gt;}_share_interrupt</td>
<td>Performs processing in response to the TIM$n$ channel$m$ interrupt.</td>
</tr>
</tbody>
</table>
R_\text{<Config_TIMn_m>_Create}

Performs initialization necessary to control the TIM gated periodic sampling mode.

Remark This API function is called by \text{R_Systeminit} function.

\textbf{[Syntax]}

\begin{verbatim}
void R_\text{<Config_TIMn_m>_Create}(void);
\end{verbatim}

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

\textbf{[Argument(s)]}
None.

\textbf{[Return value]}
None.
R_<Config_TIMn_m>_Start

Starts periodic sampling for channel \( m \) in unit \( n \).

[Syntax]

```c
void R_<Config_TIMn_m>_Start(void);
```

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

[Argument(s)]

None.

[Return value]

None.
**R_\text{\textless} \text{Config\_TIMn\_m}\text{\textgreater}\_Stop**

Stops periodic sampling for channel \(m\) in unit \(n\).

**[Syntax]**

\[
\text{void R_\text{\textless} \text{Config\_TIMn\_m}\text{\textgreater}\_Stop(\text{void});}
\]

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TIMn_m>_UpdateElapsedClock

Updates elapsed clock number.

[Syntax]

```c
void R_<Config_TIMn_m>_UpdateElapsedClock (uint32_t number);
```

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>I</td>
<td>Uint32_t number</td>
<td>Elapsed clock number</td>
</tr>
</tbody>
</table>

[Return value]

None.
**R_<Config_TIMn_m>_Software_Reset**

Stops channel operation and reset channel registers.

**[Syntax]**

```c
void R_<Config_TIMn_m>_Software_Reset(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_<_Config_TIM_n_m>_NEWVAL_TriggerOn

Trigger NEWVAL bit in TIMn.CHm_IRQ_NOTIFY register by software.

[Syntax]

```c
void R_<_Config_TIM_n_m>_NEWVAL_TriggerOn(void);
```

Remark

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

[Argument(s)]

None.

[Return value]

None.
**R_<Config_TIMn_m>_ECNTOFL_TriggerOn**

Trigger ECNTOFL bit in TIMn_CHm_IRQ_NOTIFY register by software.

**[Syntax]**

```c
void R_<Config_TIMn_m>_ECNTOFL_TriggerOn(void);
```

**Remark**

n is the unit number, m is the channel number

**[Argument(s)]**

None.

**[Return value]**

None.
**R_{<Config\_TIMn\_m>\_CNTOFL\_TriggerOn}**

Trigger CNTOFL bit in TIMn_CHm_IRQ_NOTIFY register by software.

**[Syntax]**

```c
void R_{<Config\_TIMn\_m>\_CNTOFL\_TriggerOn}(void);
```

**Remark**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_TIMn_m>_GPROFL_TriggerOn**

Trigger GPROFL bit in TIMn_CHm_IRQ_NOTIFY register by software.

**[Syntax]**

```c
void R_<Config_TIMn_m>_GPROFL_TriggerOn(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TIMn_m>_TODETOFL_TriggerOn

Trigger TODETOFL bit in TIMn_CHm_IRQ_NOTIFY register by software.

[Syntax]

```c
void R_<Config_TIMn_m>_TODETOFL_TriggerOn(void);
```

Remark  

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

[Argument(s)]

None.

[Return value]

None.
R_<Config_TIMn_m>_GLITCHDETOFL_TriggerOn

Trigger GLITCHDETOFL bit in TIMn_CHm_IRQ_NOTIFY register by software

[Syntax]

```
void R_<Config_TIMn_m>_GLITCHDETOFL_TriggerOn(void);
```

Remark   

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

[Argument(s)]

None.

[Return value]

None.
**R_{<\text{Config\_TIM}_{n\_m}>\_Create\_UserInit**

Performs user-defined initialization relating to the TIM gated periodic sampling mode.

*Remark*  This API function is called as the \texttt{R_{<\text{Config\_TIM}_{n\_m}>\_Create} callback routine.

**[Syntax]**

\begin{verbatim}
void R_{<\text{Config\_TIM}_{n\_m}>\_Create\_UserInit(void);
\end{verbatim}

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_TIMn_m>_Callback_GTM_Error**

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

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

**[Syntax]**

```
void r_<Config_TIMn_m>_Callback_GTM_Error(void);
```

**[Argument(s)]**

None.

**[Return value]**

None.
r_<Config_TIMn_m>_share_interrupt

Performs processing in response to the TIMn channelm interrupt.

[Syntax]

void r_<Config_TIMn_m>_share_interrupt(void);

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

[Argument(s)]
None.

[Return value]
None.
### 3.2.44 TIM Input Prescaler Mode

Below is a list of API functions output by the Code Generator for TIM input prescaler mode use.

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_Create</td>
<td>Performs initialization necessary to control the TIM input prescaler mode</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_Start</td>
<td>Starts the count for channel m in unit n</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_Stop</td>
<td>Stops the count for channel m in unit n</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_SetDetectedEdge</td>
<td>Sets detected edge number</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_Software_Reset</td>
<td>Stops channel operation and reset channel registers</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_NEWVAL_TriggerOn</td>
<td>Trigger NEWVAL bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_ECNTOFL_TriggerOn</td>
<td>Trigger ECNTOFL bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_CNTOFL_TriggerOn</td>
<td>Trigger CNTOFL bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_GPROFL_TriggerOn</td>
<td>Trigger GPROFL bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_TODETOFL_TriggerOn</td>
<td>Trigger TODETOFL bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_GLITCHDETOFL_TriggerOn</td>
<td>Trigger GLITCHDET bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the TIM input prescaler mode</td>
</tr>
<tr>
<td>r_&lt;Config_TIMn_m&gt;_Callback_GTM_Error</td>
<td>Processing is performed according to the occurrence of a count error interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_TIMn_m&gt;_share_interrupt</td>
<td>Performs processing in response to the TIMn channelm interrupt</td>
</tr>
</tbody>
</table>
R_<Config_TIMn_m>_Create

Performs initialization necessary to control the TIM input prescaler mode.

Remark This API function is called by R_Systeminit function.

[Syntax]

void R_<Config_TIMn_m>_Create(void);

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

[Argument(s)]

None.

[Return value]

None.
R_<Config.TIMn_m>_Start

Starts the count for channel \( m \) in unit \( n \).

[Syntax]

```c
void R_<Config.TIMn_m>_Start(void);
```

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

[Argument(s)]

None.

[Return value]

None.
R_<_conf_TIMn_m>_Stop

Stops the count for channel \( m \) in unit \( n \).

**[Syntax]**

```c
void R_<conf_TIMn_m>_Stop(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_\text{Config\_TIMn\_m}\_SetDetectedEdge

Sets detected edge number.

[Syntax]

\begin{verbatim}
void \text{R_\text{Config\_TIMn\_m}\_SetDetectedEdge (uint32\_t number)};
\end{verbatim}

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

[Argument(s)]

\begin{tabular}{|c|c|}
\hline
I/O & Argument(s) & Description \\
\hline
I & Uint32\_t number & Detected edge number \\
\hline
\end{tabular}

[Return value]

None.
**R_{<Config_TIMn_m>_Software_Reset}**

Stops channel operation and reset channel registers.

**[Syntax]**

```c
void R_{<Config_TIMn_m>_Software_Reset(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TIMn_m>_NEWVAL_TriggerOn

Trigger NEWVAL bit in TIMn_CHm_IRQ_NOTIFY register by software.

[Syntax]

void R_<Config_TIMn_m>_NEWVAL_TriggerOn(void);

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

[Argument(s)]

None.

[Return value]

None.
R_<Config_TIMn_m>_ECNTOFL_TriggerOn

Trigger ECNTOFL bit in TIMn_CHm_IRQ_NOTIFY register by software.

[Syntax]

void R_<Config_TIMn_m>_ECNTOFL_TriggerOn(void);

Remark

n is the unit number, m is the channel number

[Argument(s)]

None.

[Return value]

None.
R_<Config_TIMn_m>_CNTOFL_TriggerOn

Trigger CNTOFL bit in TIMn_CHm_IRQ_NOTIFY register by software.

[Syntax]

```c
void R_<Config_TIMn_m>_CNTOFL_TriggerOn(void);
```

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

[Argument(s)]
None.

[Return value]
None.
R_<Config.TIM_n_m>_GPROFL_TriggerOn

Trigger GPROFL bit in TIMn.CHmIRQ.NOTIFY register by software.

[Syntax]

```c
void R_<Config_TIM_n_m>_GPROFL_TriggerOn(void);
```

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

[Argument(s)]

None.

[Return value]

None.
**R_<Config_TIM_n_m>_TODETOFL_TriggerOn**

Trigger TODETOFL bit in TIMn_CHm_IRQ_NOTIFY register by software.

**[Syntax]**

```c
void R_<Config_TIM_n_m>_TODETOFL_TriggerOn(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_TIN_{m}>_GLITCHDETOFL_TriggerOn**

Trigger GLITCHDETOFL bit in TIM_{n}CH_{m}_IRQ_NOTIFY register by software

**[Syntax]**

```c
void R_<Config_TIN_{m}>_GLITCHDETOFL_TriggerOn(void);
```

Remark  

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_TIMn_m>_Create_UserInit**

Performs user-defined initialization relating to the TIM input prescaler mode.

Remark  This API functions is called as the R_<Config_TIMn_m>_Create callback routine.

**[Syntax]**

```c
void R_<Config_TIMn_m>_Create_UserInit(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

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

Remark This API functions is called as the \texttt{r\_gtm\_error\_interrupt} callback routine.

**[Syntax]**

\begin{verbatim}
void r\_<Config\_TIMn\_m>\_Callback\_GTM\_Error(void);
\end{verbatim}

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_\langle Config_TIMn_m\rangle_share_interrupt**

Performs processing in response to the TIMn channelm interrupt.

**Syntax**

```c
void r_\langle Config_TIMn_m\rangle_share_interrupt(void);
```

Remark

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

**Argument(s)**

None.

**Return value**

None.
### 3.2.45 TIM Input Event Mode

Below is a list of API functions output by the Code Generator for TIM input event mode use.

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_Create</td>
<td>Performs initialization necessary to control the TIM input event mode</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_Start</td>
<td>Starts the count for channel m in unit n</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_Stop</td>
<td>Stops the count for channel m in unit n</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_Software_Reset</td>
<td>Stops channel operation and reset channel registers</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_NEWVAL_TriggerOn</td>
<td>Trigger NEWVAL bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_ECNTOFL_TriggerOn</td>
<td>Trigger ECNTOFL bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_CNTOFL_TriggerOn</td>
<td>Trigger CNTOFL bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_GPROFL_TriggerOn</td>
<td>Trigger GPROFL bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_TODETOFL_TriggerOn</td>
<td>Trigger TODETOFL bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_GLITCHDETOFL_TriggerOn</td>
<td>Trigger GLITCHDET bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the TIM input event mode</td>
</tr>
<tr>
<td>r_&lt;Config_TIMn_m&gt;_Callback_GTM_Error</td>
<td>Processing is performed according to the occurrence of a count error interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_TIMn_m&gt;_share_interrupt</td>
<td>Performs processing in response to the TIMn channelm interrupt</td>
</tr>
</tbody>
</table>
R_<Config_TIMn_m>_Create

Performs initialization necessary to control the TIM input event mode.

Remark  This API function is called by R_Systeminit function.

[Syntax]

```c
void R_<Config_TIMn_m>_Create(void);
```

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

[Argument(s)]

None.

[Return value]

None.
**R_\langle Config_TIMn_m\rangle_Start**

Starts the count for channel \(m\) in unit \(In\).

**[Syntax]**

```c
void R_\langle Config_TIMn_m\rangle_Start(void);
```

**Remark**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_TIMn_m>_Stop**

Stops the count for channel \(m\) in unit \(n\).

**[Syntax]**

```c
void R_<Config_TIMn_m>_Stop(void);
```

**Remark**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TIMn_m>_Software_Reset

Stops channel operation and reset channel registers.

[Syntax]

```c
void R_<Config_TIMn_m>_Software_Reset(void);
```

Remark  

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

[Argument(s)]

None.

[Return value]

None.
**R_<Config_TIMn_m>_NEWVAL_TriggerOn**

Trigger NEWVAL bit in TIMn.CHm_IRQ_NOTIFY register by software.

**[Syntax]**

```c
void R_<Config_TIMn_m>_NEWVAL_TriggerOn(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**Function**: R_<Config_TIMn_m>_ECNTOFL_TriggerOn

Trigger ECNTOFL bit in TIMn_CHm_IRQ_NOTIFY register by software.

**Syntax**

```c
void R_<Config_TIMn_m>_ECNTOFL_TriggerOn(void);
```

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

**Argument(s)**

None.

**Return value**

None.
**[R_</Config_TIM\_n\_m>_]_CNTOFL_TriggerOn**

Trigger CNTOFL bit in TIM\_n\_CH\_m\_IRQ\_NOTIFY register by software.

**[Syntax]**

```c
void R_<Config_TIM\_n\_m>_CNTOFL_TriggerOn(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TIMn_m>_GPROFL_TriggerOn

Trigger GPROFL bit in TIMn_CHm_IRQ_NOTIFY register by software.

**[Syntax]**

```c
void R_<Config_TIMn_m>_GPROFL_TriggerOn(void);
```

Remark    

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TIMn_m>_TODETOFL_TriggerOn

Trigger TODETOFL bit in TIMn_CHm_IRQ_NOTIFY register by software.

[Syntax]

```c
void R_<Config_TIMn_m>_TODETOFL_TriggerOn(void);
```

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

[Argument(s)]
None.

[Return value]
None.
R\_<Config\_TIM_n\_m>_GLITCHDETOFL\_TriggerOn

Trigger GLITCHDETOFL bit in TIMn\_CHm\_IRQ\_NOTIFY register by software

**[Syntax]**

```c
void R\_<Config\_TIM_n\_m>_GLITCHDETOFL\_TriggerOn(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_TIMn_m>_Create_UserInit**

Performs user-defined initialization relating to the TIM input event mode.

Remark This API function is called as the **R_<Config_TIMn_m>_Create** callback routine.

**[Syntax]**

```c
void R_<Config_TIMn_m>_Create_UserInit(void);
```

Remark  

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_TIMn_m>_Callback_GTM_Error**

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

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

**[Syntax]**

```c
void r_<Config_TIMn_m>_Callback_GTM_Error(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_TIMn_m>_share_interrupt**

Performs processing in response to the TIM\(n\) channel\(m\) interrupt.

**[Syntax]**

```c
void r_<Config_TIMn_m>_share_interrupt(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.46 TIM Pulse Integration Mode

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

Table 3.46 API Functions: [TIM pulse integration mode]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_Create</td>
<td>Performs initialization necessary to control the TIM pulse integration mode</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_Start</td>
<td>Starts the count for channel m in unit n</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_Stop</td>
<td>Stops the count for channel m in unit n</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_Software_Reset</td>
<td>Stops channel operation and reset channel registers</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_NEWVAL_TriggerOn</td>
<td>Trigger NEWVAL bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_ECNTOFL_TriggerOn</td>
<td>Trigger ECNTOFL bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_CNTOFL_TriggerOn</td>
<td>Trigger CNTOFL bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_GPROFL_TriggerOn</td>
<td>Trigger GPROFL bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_TODETOFL_TriggerOn</td>
<td>Trigger TODETOFL bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_GLITCHDETOFL_TriggerOn</td>
<td>Trigger GLITCHDET bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the TIM pulse integration mode</td>
</tr>
<tr>
<td>r_&lt;Config_TIMn_m&gt;_Callback_GTM_Error</td>
<td>Processing is performed according to the occurrence of a count error interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_TIMn_m&gt;_share_interrupt</td>
<td>Performs processing in response to the TIMn channelm interrupt</td>
</tr>
</tbody>
</table>
R_<Config_TIMn_m>_Create

Performs initialization necessary to control the TIM pulse integration mode.

Remark  This API function is called by R_Systeminit function.

[Syntax]

void R_<Config_TIMn_m>_Create(void);

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

[Argument(s)]

None.

[Return value]

None.
**R_<Config_TIMn_m>_Start**

Starts the count for channel \( m \) in unit \( n \).

**[Syntax]**

\[
\text{void } \text{R_<Config_TIMn_m>_Start(\text{void});}
\]

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_TIMn_m>_Stop**

Stops the count for channel \(m\) in unit \(n\).

**[Syntax]**

```
void R_<Config_TIMn_m>_Stop(void);
```

**Remark**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config.TIMn_m>_Software_Reset

Stops channel operation and reset channel registers.

**[Syntax]**

```c
void R_<Config.TIMn_m>_Software_Reset(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TIMn_m>_NEWVAL_TriggerOn

Trigger NEWVAL bit in TIMn_CHm_IRQ_NOTIFY register by software.

[Syntax]

```c
void R_<Config_TIMn_m>_NEWVAL_TriggerOn(void);
```

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

[Argument(s)]

None.

[Return value]

None.
### R_<Config_TIMn_m>_ECNTOFL_TriggerOn

Trigger ECNTOFL bit in TIMn_CHm_IRQ_NOTIFY register by software.

#### [Syntax]
```c
void R_<Config_TIMn_m>_ECNTOFL_TriggerOn(void);
```

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

#### [Argument(s)]
- None.

#### [Return value]
- None.
R_<Config_TIMn_m>_CNTOFL_TriggerOn

Trigger CNTOFL bit in TIMn_CHm_IRQ_NOTIFY register by software.

[Syntax]

void R_<Config_TIMn_m>_CNTOFL_TriggerOn(void);

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

[Argument(s)]

None.

[Return value]

None.
R_\text{<Config\_TIM\_n\_m>\_GPROFL\_TriggerOn}

Trigger GPROFL bit in TIMn\_CHm\_IRQ\_NOTIFY register by software.

**[Syntax]**

```c
void R_\text{<Config\_TIM\_n\_m>\_GPROFL\_TriggerOn}()
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TIMn_m>_TODETOFL_TriggerOn

Trigger TODETOFL bit in TIMn_CHm_IRQ_NOTIFY register by software.

[Syntax]

void R_<Config_TIMn_m>_TODETOFL_TriggerOn(void);

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

[Argument(s)]

None.

[Return value]

None.
R_<Config_TIMn_m>_GLITCHDETOFL_TriggerOn

Trigger GLITCHDETOFL bit in TIMn_CHm_IRQ_NOTIFY register by software

[Syntax]

```c
void R_<Config_TIMn_m>_GLITCHDETOFL_TriggerOn(void);
```

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

[Argument(s)]

None.

[Return value]

None.
**R_<Config_TIMn_m>_Create_UserInit**

Performs user-defined initialization relating to the TIM pulse integration mode.

**Remark**  This API function is called as the **R_<Config_TIMn_m>_Create** callback routine.

**[Syntax]**

```c
void R_<Config_TIMn_m>_Create_UserInit(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

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

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

**Syntax**

```c
void _r<Config_TIMn_m>_Callback_GTM_Error(void);
```

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

**Argument(s)**

None.

**Return value**

None.
**r_<Config_TIMn_m>_share_interrupt**

Performs processing in response to the TIMn channelm interrupt.

**[Syntax]**

```c
void r_<Config_TIMn_m>_share_interrupt(void);
```

Remark  

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

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.47 TIM PWM Measurement Mode

Below is a list of API functions output by the Code Generator for TIM pwm measurement mode use.

Table 3.47 API Functions: [TIM pwm measurement mode]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_Create</td>
<td>Performs initialization necessary to control the TIM pwm measurement mode</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_Start</td>
<td>Starts the count for channel (m) in unit (n)</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_Stop</td>
<td>Stops the count for channel (m) in unit (n)</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_Software_Reset</td>
<td>Stops channel operation and reset channel registers</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_NEWVAL_TriggerOn</td>
<td>Trigger NEWVAL bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_ECNTOFL_TriggerOn</td>
<td>Trigger ECNTOFL bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_CNTOFL_TriggerOn</td>
<td>Trigger CNTOFL bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_GPROFL_TriggerOn</td>
<td>Trigger GPROFL bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_TODETOFL_TriggerOn</td>
<td>Trigger TODETOFL bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_GLITCHDETOFL_TriggerOn</td>
<td>Trigger GLITCHDET bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the TIM pwm measurement mode</td>
</tr>
<tr>
<td>r_&lt;Config_TIMn_m&gt;_Callback_GTM_Error</td>
<td>Processing is performed according to the occurrence of a count error interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_TIMn_m&gt;_share_interrupt</td>
<td>Performs processing in response to the TIM(n) channel(m) interrupt</td>
</tr>
</tbody>
</table>
**R_<Config_TIMn_m>_Create**

Performs initialization necessary to control the TIM PWM measurement mode.

**Remark**
This API function is called by `R_Systeminit` function.

**[Syntax]**

```c
void R_<Config_TIMn_m>_Create(void);
```

**Remark**

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TIMn_m>_Start

Starts the count for channel $m$ in unit $n$.

[Syntax]

```c
void R_<Config_TIMn_m>_Start(void);
```

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

[Argument(s)]

None.

[Return value]

None.
R_<Config.TIMn_m>_Stop

Stops the count for channel \( m \) in unit \( n \).

**Syntax**

```c
void R_<Config.TIMn_m>_Stop(void);
```

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

**Argument(s)**

None.

**Return value**

None.
R_<Config_TIMn_m>_Software_Reset

Stops channel operation and reset channel registers.

[Syntax]

```c
void R_<Config_TIMn_m>_Software_Reset(void);
```

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

[Argument(s)]

None.

[Return value]

None.
R_<Config_TIMn_m>_NEWVAL_TriggerOn

Trigger NEWVAL bit in TIMn_CHm_IRQ_NOTIFY register by software.

[Syntax]

```c
void R_<Config_TIMn_m>_NEWVAL_TriggerOn(void);
```

Remark

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

[Argument(s)]

None.

[Return value]

None.
Trigger ECNTOFL bit in TIM_nChm_IRQ_NOTIFY register by software.

**Syntax**

```c
void R_<Config_TIMn_m>_ECNTOFL_TriggerOn(void);
```

**Remark**

- `n` is the unit number,
- `m` is the channel number

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TIMn_m>_CNTOFL_TriggerOn

Trigger CNTOFL bit in TIMn_CHm_IRQHandler register by software.

[Syntax]

```c
void R_<Config_TIMn_m>_CNTOFL_TriggerOn(void);
```

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

[Argument(s)]

None.

[Return value]

None.
R_<Config_TIMn_m>_GPROFL_TriggerOn

Trigger GPROFL bit in TIMn_CHm_IRQ_NOTIFY register by software.

[Syntax]

void R_<Config_TIMn_m>_GPROFL_TriggerOn(void);

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

[Argument(s)]

None.

[Return value]

None.
**R_<Config_TIMn_m>_TODETOFL_TriggerOn**

Trigger TODETOFL bit in TIMn_CHm_IRQ_NOTIFY register by software.

**[Syntax]**

```c
void R_<Config_TIMn_m>_TODETOFL_TriggerOn(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R\_<Config\_TIMn\_m>_GLITCHDETOFL\_TriggerOn**

Trigger GLITCHDETOFL bit in TIMn\_CH\_m\_IRQ\_NOTIFY register by software

**[Syntax]**

```c
void R\_<Config\_TIMn\_m>_GLITCHDETOFL\_TriggerOn(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_{<Config_TIM\_n\_m>\_Create\_UserInit**

Performs user-defined initialization relating to the TIM PWM measurement mode.

**Remark** This API function is called as the **R_{<Config_TIM\_n\_m>\_Create** callback routine.

**[Syntax]**

```c
void R_{<Config_TIM\_n\_m>\_Create\_UserInit(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
### r_\langle Config\_TIMn\_m\rangle\_Callback\_GTM\_Error

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

**Remark**  
This API function is called as the \texttt{r_gtm_error_interrupt} callback routine.

#### [Syntax]

```c
void r_\langle Config\_TIMn\_m\rangle\_Callback\_GTM\_Error(void);
```

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

#### [Argument(s)]

None.

#### [Return value]

None.
**r_<Config_TIMn_m>_share_interrupt**

Performs processing in response to the TIMn channelm interrupt.

**[Syntax]**

```c
void r_<Config_TIMn_m>_share_interrupt(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
### 3.48 TIM Serial Shift Mode

Below is a list of API functions output by the Code Generator for TIM serial shift mode use.

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_Create</td>
<td>Performs initialization necessary to control the TIM serial shift mode.</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_Start</td>
<td>Starts serial shift for channel m in unit n</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_Stop</td>
<td>Stops serial shift for channel m in unit n</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_UpdateElapsedClock</td>
<td>Updates elapsed clock number.</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_UpdateShiftClock</td>
<td>Updates shift clock number</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_UpdateCaptureSource</td>
<td>Updates capture source</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_UpdateTssmOut</td>
<td>Updates tssm out signal</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_SetGPR0AsShadowRegister</td>
<td>Sets TIMn_CHm_GPR0 as a shadow register for TIMn_CHm_CNT</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_Software_Reset</td>
<td>Stops channel operation and reset channel registers</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_NEWVAL_TriggerOn</td>
<td>Trigger NEWVAL bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_ECNTOFL_TriggerOn</td>
<td>Trigger ECNTOFL bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_CNTOFL_TriggerOn</td>
<td>Trigger CNTOFL bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_GPROFL_TriggerOn</td>
<td>Trigger GPROFL bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_TODETOFL_TriggerOn</td>
<td>Trigger TODETOFL bit in TIMn_CHmIRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_GLITCHDETOFL_TriggerOn</td>
<td>Trigger GLITCHDET bit in TIMn_CHm_IRQ_NOTIFY register by software</td>
</tr>
<tr>
<td>R_&lt;Config_TIMn_m&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the TIM serial shift mode.</td>
</tr>
<tr>
<td>r_&lt;Config_TIMn_m&gt;_Callback_GTM_Error</td>
<td>Processing is performed according to the occurrence of a count error interrupt.</td>
</tr>
<tr>
<td>r_&lt;Config_TIMn_m&gt;_share_interrupt</td>
<td>Performs processing in response to the TIMn channel m interrupt.</td>
</tr>
</tbody>
</table>
### R_<Config_TIMn_m>_Create

Performs initialization necessary to control the TIM serial shift mode.

**Remark** This API function is called by `R_Systeminit` function.

#### [Syntax]

```c
void R_<Config_TIMn_m>_Create(void);
```

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

#### [Argument(s)]

None.

#### [Return value]

None.
**R_<Config_TIMn_m>_Start**

Starts the count for channel \( m \) in unit \( n \).

**[Syntax]**

```c
void R_<Config_TIMn_m>_Start(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TIMn_m>_Stop

Stops the count for channel \( m \) in unit \( n \).

**Syntax**

```c
void R_<Config_TIMn_m>_Stop(void);
```

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

**Argument(s)**

None.

**Return value**

None.
### R_<Config_TIMn_m>_UpdateElapsedClock

Updates elapsed clock number.

**[Syntax]**

```c
void R_<Config_TIMn_m>_UpdateElapsedClock(uint8_t number);
```

**Remark**  
\( \text{n} \) is the unit number, \( \text{m} \) 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>I</td>
<td>Uint8_t number</td>
<td>Elapsed clock number</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
R_<_Config_TIMn_m>_UpdateShiftClock

Updates shift clock number.

[Syntax]

```c
void R_<_Config_TIMn_m>_UpdateShiftClock(uint8_t number);
```

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>I</td>
<td>Uint8_t number</td>
<td>Shift clock number</td>
</tr>
</tbody>
</table>

[Return value]

None.
**R_{<\text{Config\_TIMn\_m}>\_UpdateCaptureSource}**

Updates capture source.

**[Syntax]**

```c
void R_{<\text{Config\_TIMn\_m}>\_UpdateCaptureSource}(uint8_t number);
```

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>I</td>
<td>Uint8_t number</td>
<td>Capture source</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
R_<Config_TIMn_m>_UpdateTssmOut

Updates tssm out.

[Syntax]

void R_<Config_TIMn_m>_UpdateTssmOut (uint8_t number);

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>I</td>
<td>Uint8_t number;</td>
<td>Tssm out signal</td>
</tr>
</tbody>
</table>

[Return value]

None.
R_{Config\_TIM_{n\_m}}\_SetGPR0AsShadowRegister

Sets TIM_{n\_CH_{m\_GPR0}} as a shadow register for TIM_{n\_CH_{m\_CNT}}.

[Syntax]

\[
\text{void } \text{R}_{<\text{Config\_TIM}_{n\_m}>}\_\text{SetGPR0AsShadowRegister(char number);}\]

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>I</td>
<td>char number</td>
<td>Sets whether to use GPR0 as the shadow register of the CNT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>register</td>
</tr>
</tbody>
</table>

[Return value]

None.
**R_<Config_TIMn_m>_Software_Reset**

Stops channel operation and reset channel registers.

**[Syntax]**

```c
void R_<Config_TIMn_m>_Software_Reset(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.


**R\_<Config\_TIM\_n\_m>\_NEWVAL\_TriggerOn**

Trigger NEWVAL bit in TIMn\_CHm\_IRQ\_NOTIFY register by software.

**[Syntax]**

```c
void R\_<Config\_TIM\_n\_m>\_NEWVAL\_TriggerOn(void);
```

**Remark**

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

**[Argument(s)]**

None.

**[Return value]**

None.
Trigger ECNTOFL bit in TIMn.CHm_IRQ_NOTIFY register by software.

**Syntax**

```c
void R_Config_TIMn_m_ECNTOFL_TriggerOn(void);
```

Remark

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

**Argument(s)**

None.

**Return value**

None.
**R_\langle Config\_TIM_n\_m\rangle\_CNTOFL\_TriggerOn**

Trigger CNTOFL bit in TIMn\_CHm\_IRQ\_NOTIFY register by software.

**[Syntax]**

```c
void R_\langle Config\_TIM_n\_m\rangle\_CNTOFL\_TriggerOn(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_\text{<Config\_TIM}\_n\_m\text{>_GPROFL\_TriggerOn}

Trigger GPROFL bit in TIM\_n\_CH\_m\_IRQ\_NOTIFY register by software.

[Syntax]

\begin{verbatim}
void R_\text{<Config\_TIM}\_n\_m\text{>_GPROFL\_TriggerOn}(void);
\end{verbatim}

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

[Argument(s)]

None.

[Return value]

None.
**R\_<Config\_TIM\_m>\_TODETOFL\_TriggerOn**

Trigger TODETOFL bit in TIMn\_CHm\_IRQ\_NOTIFY register by software.

**[Syntax]**

```
void R\_<Config\_TIM\_m>\_TODETOFL\_TriggerOn(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_TIMn_m>_GLITCHDETOFL_TriggerOn

Trigger GLITCHDETOFL bit in TIMn_CHm_IRQ_NOTIFY register by software

[Syntax]

void R_<Config_TIMn_m>_GLITCHDETOFL_TriggerOn(void);

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

[Argument(s)]

None.

[Return value]

None.
**R_<Config_TIMn_m>_Create_UserInit**

Performs user-defined initialization relating to the TIM serial shift mode.

**Remark**
This API function is called as the `R_<Config_TIMn_m>_Create` callback routine.

**[Syntax]**

```c
void R_<Config_TIMn_m>_Create_UserInit(void);
```

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

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

**[Return value]**
None.
r_<Config_TIMn_m>_Callback_GTM_Error

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

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

[Syntax]
void r_<Config_TIMn_m>_Callback_GTM_Error(void);

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

[Argument(s)]
None.

[Return value]
None.
**r_<Config_TIMn_m>_share_interrupt**

Performs processing in response to the TIMn channelm interrupt.

**[Syntax]**

```
void r_<Config_TIMn_m>_share_interrupt(void);
```

**Remark**

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

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.49 ATOM Signal Output Mode Compare

Below is a list of API functions output by the Code Generator for ATOM signal output mode compare use.

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_ATOMn_m&gt;_Create</td>
<td>Performs initialization necessary to control the ATOM signal output mode compare</td>
</tr>
<tr>
<td>R_&lt;Config_ATOMn_m&gt;_Start</td>
<td>Starts ATOMnm signal output mode compare</td>
</tr>
<tr>
<td>R_&lt;Config_ATOMn_m&gt;_Stop</td>
<td>Stop ATOMnm signal output mode compare</td>
</tr>
<tr>
<td>R_&lt;Config_ATOMn_m&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the ATOM signal output mode compare functions</td>
</tr>
<tr>
<td>r_&lt;Config_ATOMn_m&gt;_Callback_Shared_IRQk</td>
<td>Performs processing in response to the ATOMnm shared interrupt</td>
</tr>
</tbody>
</table>
**R_<Config_ATOMn_m>_Create**

Performs initialization necessary to control the ATOM signal output mode compare.

**Remark** This API function is called by **R_Systeminit** function.

**[Syntax]**

```c
void R_<Config_ATOMn_m>_Create(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_\text{<Config_ATOMn_m>_Start}**

Starts ATOMnm signal output mode compare.

**[Syntax]**

```c
void R_\text{<Config_ATOMn_m>_Start}(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_ATOMn_m>_Stop

Stop ATOMnm signal output mode compare.

[Syntax]

```c
void R_<Config_ATOMn_m>_Stop(void);
```

Remark

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

[Argument(s)]

None.

[Return value]

None.
Performs user-defined initialization relating to the ATOM signal output mode compare functions.

Remark This API function is called as the $R_{<\text{Config}\_\text{ATOM}_n\_m>\_\text{Create}}$ callback routine.

**[Syntax]**

```c
void $R_{<\text{Config}\_\text{ATOM}_n\_m>\_\text{Create}\_\text{UserInit}}$(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
### r\_<Config_ATOMn_m>_Callback_Shared_IRQk

Performs processing in response to the ATOMnm shared interrupt.

**Remark**  This API function is called by \texttt{r\_atomn\_irqk\_interrupt} function.

**[Syntax]**

```c
void r\_<Config_ATOMn_m>_Callback_Shared_IRQk (void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.50 ATOM Signal Output Mode Immediate

Below is a list of API functions output by the Code Generator for ATOM signal output mode immediate use.

Table 3.50 API Functions: [ATOM Signal Output Mode Immediate]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_ATOMn_m&gt;_Create</td>
<td>Performs initialization necessary to control the ATOM signal output mode immediate</td>
</tr>
<tr>
<td>R_&lt;Config_ATOMn_m&gt;_Start</td>
<td>Starts ATOMnm signal output mode immediate</td>
</tr>
<tr>
<td>R_&lt;Config_ATOMn_m&gt;_Stop</td>
<td>Stop ATOMnm signal output mode immediate</td>
</tr>
<tr>
<td>R_&lt;Config_ATOMn_m&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the ATOM signal output mode immediate functions</td>
</tr>
</tbody>
</table>
**R<_Config_ATOMn_m>_Create**

Performs initialization necessary to control the ATOM signal output mode immediate.

**Remark**
This API function is called by **R_Systeminit** function.

**[Syntax]**

```c
void R<_Config_ATOMn_m>_Create(void);
```

**Remark**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_ATOMn_m>_Start**

Starts ATOMnm signal output mode immediate.

**[Syntax]**

```c
void R_<Config_ATOMn_m>_Start(void);
```

Remark  

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R<_Config_ATOMn_m>_Stop**

Stop ATOMnm signal output mode immediate.

**[Syntax]**

```c
void R<_Config_ATOMn_m>_Stop(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_\text{<Config\_ATOMn\_m>\_Create\_UserInit}**

Performs user-defined initialization relating to the ATOM signal output mode immediate functions.

**Remark** This API function is called as the \text{R\_<Config\_ATOMn\_m>\_Create} callback routine.

**[Syntax]**

\begin{verbatim}
void R_\text{<Config\_ATOMn\_m>\_Create\_UserInit}(void);
\end{verbatim}

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

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

**[Return value]**
None.
3.2.51 ATOM Signal Output Mode PWM

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

Table 3.51 API Functions: [ATOM Signal Output Mode PWM]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_ATOMn_m&gt;_Create</td>
<td>Performs initialization necessary to control the ATOM signal output mode PWM</td>
</tr>
<tr>
<td>R_&lt;Config_ATOMn_m&gt;_Start</td>
<td>Starts ATOMnm signal output mode PWM</td>
</tr>
<tr>
<td>R_&lt;Config_ATOMn_m&gt;_Stop</td>
<td>Stop ATOMnm signal output mode PWM</td>
</tr>
<tr>
<td>r_&lt;Config_ATOMn_m&gt;_Callback_Shared_IRQk</td>
<td>Performs user-defined initialization relating to the ATOM signal output mode PWM functions</td>
</tr>
<tr>
<td>r_&lt;Config_ATOMn_m&gt;_callback_shared_interrupt</td>
<td>Performs processing in response to the ATOMnm shared interrupt</td>
</tr>
</tbody>
</table>
R_<Config_ATOMn_m>_Create

Performs initialization necessary to control the ATOM signal output mode PWM.

Remark  This API function is called by R_Systeminit function.

[Syntax]

void R_<Config_ATOMn_m>_Create(void);

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

[Argument(s)]

None.

[Return value]

None.
**R_<config_ATOMn_m>_Start**

Starts ATOMnm signal output mode PWM.

**[Syntax]**

```c
void R_<config_ATOMn_m>_Start(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_ATOMn_m>_Stop
Stop ATOMnm signal output mode PWM.

[Syntax]

void R_<ConfigATOMn_m>_Stop(void);

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

[Argument(s)]
None.

[Return value]
None.
R_<Config_ATOMn_m>_Create_UserInit

Performs user-defined initialization relating to the ATOM signal output mode PWM functions.

Remark This API functions is called as the R_<Config_ATOMn_m>_Create callback routine.

[Syntax]

```c
void R_<Config_ATOMn_m>_Create_UserInit(void);
```

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

[Argument(s)]

None.

[Return value]

None.
**r_{<Config_ATOMn_m>}_Callback_Shared_IRQk**

Performs processing in response to the ATOMnm shared interrupt.

**Remark** This API function is called by `r_atomn_irqk_interrupt` function.

**[Syntax]**

```c
void r_{<Config_ATOMn_m>}_Callback_Shared_IRQk (void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.52 ATOM Signal Output Mode Serial

Below is a list of API functions output by the Code Generator for ATOM signal output mode serial use.

Table 3.52 API Functions: [ATOM Signal Output Mode Serial]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_{&lt;Config_ATOMn_m&gt;_Create}</td>
<td>Performs initialization necessary to control the ATOM signal</td>
</tr>
<tr>
<td></td>
<td>output mode serial</td>
</tr>
<tr>
<td>R_{&lt;Config_ATOMn_m&gt;_Start}</td>
<td>Starts ATOMnm signal output mode serial</td>
</tr>
<tr>
<td>R_{&lt;Config_ATOMn_m&gt;_Stop}</td>
<td>Stop ATOMnm signal output mode serial</td>
</tr>
<tr>
<td>R_{&lt;Config_ATOMn_m&gt;_Create_UserInit}</td>
<td>Performs user-defined initialization relating to the ATOM</td>
</tr>
<tr>
<td></td>
<td>signal output mode serial functions</td>
</tr>
<tr>
<td>r_{&lt;Config_ATOMn_m&gt;_Callback_Shared_IRQk}</td>
<td>Performs processing in response to the ATOMnm shared</td>
</tr>
<tr>
<td></td>
<td>interrupt</td>
</tr>
</tbody>
</table>
**R_<Config_ATOMn_m>_Create**

Performs initialization necessary to control the ATOM signal output mode serial.

**Remark**
This API function is called by **R_Systeminit** function.

**[Syntax]**

```c
void R_<Config_ATOMn_m>_Create(void);
```

**Remark**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_<Config_ATOMn_m>_Start**

Starts ATOMnm signal output mode serial.

**Syntax**

```c
void R_<Config_ATOMn_m>_Start(void);
```

Remark  

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

**Argument(s)**

None.

**Return value**

None.
R_<Config_ATOMn_m>_Stop

Stop ATOMnm signal output mode serial.

[Syntax]

```c
void R_<Config_ATOMn_m>_Stop(void);
```

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

[Argument(s)]

None.

[Return value]

None.
**R_<Config_ATOMn_m>_Create_UserInit**

Performs user-defined initialization relating to the ATOM signal output mode serial functions.

Remark  This API functions is called as the **R_<Config_ATOMn_m>_Create** callback routine.

**[Syntax]**

```c
void R_<Config_ATOMn_m>_Create_UserInit(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config_ATOM_n>m>_Callback_Shared_IRQk**

Performs processing in response to the ATOMnm shared interrupt.

**Remark**  This API function is called by **r_atomn_irqk_interrupt** function.

**[Syntax]**

```c
void r_<Config_ATOM_n>m>_Callback_Shared_IRQk (void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.53  Dead Time Module

Below is a list of API functions output by the Code Generator for dead time module use.

Table 3.53 API Functions: [Dead Time Module]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_CDTMn_m&gt;_Create</td>
<td>Performs initialization necessary to control the dead time module</td>
</tr>
<tr>
<td>R_&lt;Config_CDTMn_m&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to the dead time module</td>
</tr>
</tbody>
</table>
**R_<Config_CDTM_n_m>_Create**

Performs initialization necessary to control the dead time module.

**Remark**
This API function is called by `R_Systeminit` function.

**[Syntax]**

```c
void R_<Config_CDTM_n_m>_Create(void);
```

**Remark**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**R_{<Config_CDTM_n_m>_Create_UserInit}**

Performs user-defined initialization relating to the dead time module.

**Remark**  This API function is called as the $R_{<Config_CDTM_n_m>_Create}$ callback routine.

**[Syntax]**

```c
void R_{<Config_CDTM_n_m>_Create_UserInit(void);
```

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

**[Argument(s)]**

None.

**[Return value]**

None.
3.2.54 DTS Controller

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

Table 3.54 API Functions: [DTS Controller]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_&lt;Config_DTSn&gt;_Create</td>
<td>Performs initialization necessary to control the DTS functions which includes chain header channel n and chained channel</td>
</tr>
<tr>
<td>R_&lt;Config_DTSn&gt;_Start</td>
<td>Enables the DTS operation which includes chain header channel n and chained channel</td>
</tr>
<tr>
<td>R_&lt;Config_DTSn&gt;_Stop</td>
<td>Disables the DTS operation which includes chain header channel n and chained channel</td>
</tr>
<tr>
<td>R_&lt;Config_DTSn&gt;_Set_SoftwareTrigger</td>
<td>Set DTS software trigger for chain header channel n</td>
</tr>
<tr>
<td>r_&lt;Config_DTSn&gt;_Callback_Dtsg_Transfer_End</td>
<td>Performs processing in response to the DTS channel g transfer end interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_DTSn&gt;_Callback_Dtsg_Transfer_Count_Match</td>
<td>Performs processing in response to the DTS channel g transfer count match interrupt</td>
</tr>
<tr>
<td>r_&lt;Config_DTSn&gt;_Callback_PEk_Transfer_Error</td>
<td>Performs processing in response to the DTS transfer error interrupt for PEk</td>
</tr>
<tr>
<td>r_&lt;Config_DTSn&gt;_Callback_Transfer_Error</td>
<td>Performs processing in response to the DTS transfer error interrupt</td>
</tr>
<tr>
<td>R_&lt;Config_DTSn&gt;_Create_UserInit</td>
<td>Performs user-defined initialization relating to DTS controller functions</td>
</tr>
</tbody>
</table>
R<Config_DTSn>_Create

Performs initialization necessary to control the DTSn functions
Remark This API function is called by R_Systeminit function.

[Syntax]
void R<Config_DTSn>_Create(void);
Remark n is the chain header channel number.

[Argument(s)]
None.

[Return value]
None.
R_<Config_DTSn>_Start

Enables the DTS operation.

[Syntax]

```c
void R_<Config_DTSn>_Start(void);
```

Remark  

n is the chain channel number.

[Argument(s)]

None.

[Return value]

None.
R_<Config_DTSn>_Stop

Disables the DTS operation.

**[Syntax]**

```c
void R_<Config_DTSn>_Stop(void);
```

Remark  \(n\) is the chain header channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_DTSn>_Set_SoftwareTrigger

Generates the DTS channel n transfer request.

**[Syntax]**

```c
void R_<Config_DTSn>_Set_SoftwareTrigger(void);
```

Remark  

\( n \) is the chain header channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
R_<Config_DTSn>._Create_UserInit

Performs user-defined initialization relating to the DTS functions.

Remark  This API function is called as the R_<Config_DTSn>._Create callback routine.

[Syntax]

```c
void R_<Config_DTSn>._Create_UserInit(void);
```

Remark  \( n \) is the chain header channel number.

[Argument(s)]

None.

[Return value]

None.
**r_<Config_DTSn>_Callback_Dtsg_Transfer_End**

Performs processing in response to the DTS transfer end interrupt.

**Remark**
This API functions is called by `r_dtsg_transfer_end_interrupt` routine.

**[Syntax]**

```c
void r_<Config_DTSn>_Callback_Dtsg_Transfer_End(void);
```

**Remark**
`n` is the chain header channel number, `g` is interrupt group number, 31to0, 63to32, 95to64, 127to96.

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

**[Return value]**
None.
**r_<Config_DTSn>_Callback_Dtsg_Transfer_Count_Match**

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

**Remark**  This API function is called by `r_dtsg_transfer_count_match_interrupt` routine.

**[Syntax]**

```c
void r_<Config_DTSn>_Callback_Dtsg_Transfer_Count_Match(void);
```

**Remark**  

$n$ is the chain header channel number, $g$ is interrupt group number, 31to0, 63to32, 95to64, 127to96.

**[Argument(s)]**

None.

**[Return value]**

None.
**r_<Config DTSn>_Callback_PEk_Transfer_Error**

Performs processing in response to the DTS transfer error interrupt.

**Remark**  This API function is called by `r_dts_transfer_error_interrupt_pek` routine.

**[Syntax]**

```c
void r_<Config DTSn>_Callback_PEk_Transfer_Error(void)
```

**Remark**  `n` is the chain header channel number, `k` is the CPU core number.

**[Argument(s)]**

None.

**[Return value]**

None.
r_<Config_DTSn>_Callback_Transfer_Error

Performs processing in response to the DTS transfer error interrupt.

Remark  This API function is called by r_dmac_error_interrupt_pek routine.

[Syntax]

```
void r_<Config_DTSn>_Callback_Transfer_Error(void)
```

Remark  n is the chain header channel number.

[Argument(s)]

None.

[Return value]

None.
3.2.55 ADC Boundary Flag Generator

Below is a list of API functions output by the Code Generator for ADC Boundary Flag Generator use.

Table 3.11 API Functions: [ADC Boundary Flag Generator]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R..&lt;Config_ABFG&gt;._Create</td>
<td>Performs initialization necessary to control the ADC Boundary Flag Generator functions</td>
</tr>
<tr>
<td>R..&lt;Config_ABFG&gt;._Start</td>
<td>Starts ABFG operation</td>
</tr>
<tr>
<td>R..&lt;Config_ABFG&gt;._Stop</td>
<td>Stops ABFG operation</td>
</tr>
<tr>
<td>R..&lt;Config_ABFG&gt;._Create_UserInit</td>
<td>Performs user-defined initialization relating to the ADC Boundary Flag Generator functions</td>
</tr>
<tr>
<td>r.&lt;Config_ABFG&gt;._boundary_flag_pulse_w_interru</td>
<td>Performs processing in response to the interrupt request signal from AIR</td>
</tr>
</tbody>
</table>
R_\text{Config\_ABFG}\_Create

Performs initialization necessary to control the ADC Boundary Flag Generator functions.

Remark  This API function is called by \text{R\_Systeminit} function.

[Syntax] 

\text{void R\_\text{Config\_ABFG}\_Create(void);}

[Argument(s)]

None.

[Return value]

None.
R_<Config_ABFG>_Start

Starts ABFG operation.

[Syntax]

```c
void R_<Config_ABFG>_Start(void);
```

[Argument(s)]

None.

[Return value]

None.
R_<Config_ABFG>_Stop

Stops ABFG operation.

[Syntax]

```c
void R_<Config_ABFG>_Stop(void);
```

[Argument(s)]

None.

[Return value]

None.
R_<Config_ABFG>_Create_UserInit

Performs user-defined initialization relating to the ADC Boundary Flag Generator functions.

Remark This API function is called as the R_<Config_ABFG>_Create callback routine.

[Syntax]

```c
void R_<Config_ABFG>_Create_UserInit(void);
```

[Argument(s)]
None.

[Return value]
None.
r_<Config_ABFG>_boundary_flag_pulse_w_interrupt

Performs processing in response to the interrupt request signal from AIR.

[Syntax]
void r_<Config_ABFG>_boundary_flag_pulse_w_interrupt(void);

Remark w is the boundary flag number.

[Argument(s)]
None.

[Return value]
None.
Revision Record

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>All</td>
<td>First Edition issued</td>
</tr>
<tr>
<td>1.10</td>
<td>3. API FUNCTIONS</td>
<td>Table 3.1.1 API Functions: [General]: Add new APIs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Add chapter 3.2.34 to 3.2.54</td>
</tr>
<tr>
<td>1.20</td>
<td>All</td>
<td>Add &quot;&lt;&gt;&quot; for all configuration name in API name</td>
</tr>
<tr>
<td></td>
<td>2. OUTPUT FILES</td>
<td>Table 2.1 Output File List: Remove all private functions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Table 2.1 Output File List: Change the &quot;start&quot; and &quot;stop&quot; API name for Clock Divide function.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Table 2.1 Output File List: Delete duplicated API name for DMA controller function.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Table 2.1 Output File List: Add &quot;r.&lt;Config_ATOMn_m&gt;_callback_shared_interrupt&quot; for ATOM Signal Output Mode PWM and ATOM Signal Output Mode Serial function</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Table 2.1 Output File List: Change the API name in [Syntax] part.</td>
</tr>
<tr>
<td>3.00</td>
<td>3. API FUNCTIONS</td>
<td>3.2.6 Input Interval Timer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Remove the following private functions from this chapter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r.&lt;Config_TAUXn_m&gt;_disable_pek_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r.&lt;Config_TAUXn_m&gt;_enable_pek_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r.&lt;Config_TAUXn_m&gt;_disable_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r.&lt;Config_TAUXn_m&gt;_enable_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.2.7 Input Pulse Interval Measurement</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Remove the following private functions from this chapter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r.&lt;Config_TAUXn_m&gt;_disable_pek_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r.&lt;Config_TAUXn_m&gt;_enable_pek_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r.&lt;Config_TAUXn_m&gt;_disable_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r.&lt;Config_TAUXn_m&gt;_enable_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.2.8 Interval Timer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Remove the following private functions from this chapter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r.&lt;Config_TAUXn_m&gt;_disable_pek_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r.&lt;Config_TAUXn_m&gt;_enable_pek_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r.&lt;Config_TAUXn_m&gt;_disable_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r.&lt;Config_TAUXn_m&gt;_enable_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.2.9 Triangle PWM Output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Remove the following private functions from this chapter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r.&lt;Config_TAUXn_m&gt;_disable_pek_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r.&lt;Config_TAUXn_m&gt;_enable_pek_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r.&lt;Config_TAUXn_m&gt;_disable_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r.&lt;Config_TAUXn_m&gt;_enable_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.2.10 Triangle PWM Output With Dead Time</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Remove the following private functions from this chapter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r.&lt;Config_TAUXn_m&gt;_disable_pek_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r.&lt;Config_TAUXn_m&gt;_enable_pek_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r.&lt;Config_TAUXn_m&gt;_disable_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r.&lt;Config_TAUXn_m&gt;_enable_interrupt</td>
</tr>
<tr>
<td>Rev.</td>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>---------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| 1.20 | 3. API FUNCTIONS | 3.2.13 PWM Output  
Remove the following private functions from this chapter

- `r.<Config_TAUXn_m>_disable_pek_interrupt`
- `r.<Config_TAUXn_m>_enable_pek_interrupt`
- `r.<Config_TAUXn_m>_disable_interrupt`
- `r.<Config_TAUXn_m>_enable_interrupt`

3.2.17 Clock Divide  
Remove the following private functions from this chapter

- `r.<Config_TAUXn_m>_disable_pek_interrupt`
- `r.<Config_TAUXn_m>_enable_pek_interrupt`
- `r.<Config_TAUXn_m>_disable_interrupt`
- `r.<Config_TAUXn_m>_enable_interrupt`

3.2.19 Delay Count  
Remove the following private functions from this chapter

- `r.<Config_TAUXn_m>_disable_pek_interrupt`
- `r.<Config_TAUXn_m>_enable_pek_interrupt`
- `r.<Config_TAUXn_m>_disable_interrupt`
- `r.<Config_TAUXn_m>_enable_interrupt`

3.2.20 DMA Controller  
Remove the following private functions from this chapter

- `r.<Config_SDMACnm>_enable_pek_interrupt`
- `r.<Config_SDMACnm>_disable_pek_interrupt`

3.2.21 External Event Count  
Remove the following private functions from this chapter

- `r.<Config_TAUXn_m>_disable_pek_interrupt`
- `r.<Config_TAUXn_m>_enable_pek_interrupt`
- `r.<Config_TAUXn_m>_disable_interrupt`
- `r.<Config_TAUXn_m>_enable_interrupt`

3.2.22 Input Period Count Detection  
Remove the following private functions from this chapter

- `r.<Config_TAUXn_m>_disable_pek_interrupt`
- `r.<Config_TAUXn_m>_enable_pek_interrupt`
- `r.<Config_TAUXn_m>_disable_interrupt`
- `r.<Config_TAUXn_m>_enable_interrupt`

3.2.23 Input Position Detection  
Remove the following private functions from this chapter

- `r.<Config_TAUXn_m>_disable_pek_interrupt`
- `r.<Config_TAUXn_m>_enable_pek_interrupt`
- `r.<Config_TAUXn_m>_disable_interrupt`
- `r.<Config_TAUXn_m>_enable_interrupt`
<table>
<thead>
<tr>
<th>Rev.</th>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1.20 | 3. API FUNCTIONS | 3.2.24 Input Pulse Interval Judgment  
Remove the following private functions from this chapter  
r.<Config_TAUXn_m>_disable_pek_interrupt  
r.<Config_TAUXn_m>_enable_pek_interrupt  
r.<Config_TAUXn_m>_disable_interrupt  
r.<Config_TAUXn_m>_enable_interrupt |
|      |               | 3.2.25 Input Signal Width Judgment  
Remove the following private functions from this chapter  
r.<Config_TAUXn_m>_disable_pek_interrupt  
r.<Config_TAUXn_m>_enable_pek_interrupt  
r.<Config_TAUXn_m>_disable_interrupt  
r.<Config_TAUXn_m>_enable_interrupt |
|      |               | 3.2.26 Input Signal Width Measurement  
Remove the following private functions from this chapter  
r.<Config_TAUXn_m>_disable_pek_interrupt  
r.<Config_TAUXn_m>_enable_pek_interrupt  
r.<Config_TAUXn_m>_disable_interrupt  
r.<Config_TAUXn_m>_enable_interrupt |
|      |               | 3.2.28 One Pulse Output  
Remove the following private functions from this chapter  
r.<Config_TAUXn_m>_disable_pek_interrupt  
r.<Config_TAUXn_m>_enable_pek_interrupt  
r.<Config_TAUXn_m>_disable_interrupt  
r.<Config_TAUXn_m>_enable_interrupt |
|      |               | 3.2.29 One-shot Pulse Output  
Remove the following private functions from this chapter  
r.<Config_TAUXn_m>_disable_pek_interrupt  
r.<Config_TAUXn_m>_enable_pek_interrupt  
r.<Config_TAUXn_m>_disable_interrupt  
r.<Config_TAUXn_m>_enable_interrupt |
|      |               | 3.2.30 Overflow Interrupt Output (Input period count detecting)  
Remove the following private functions from this chapter  
r.<Config_TAUXn_m>_disable_pek_interrupt  
r.<Config_TAUXn_m>_enable_pek_interrupt  
r.<Config_TAUXn_m>_disable_interrupt  
r.<Config_TAUXn_m>_enable_interrupt |
|      |               | 3.2.31 Overflow Interrupt Output (Width measurement)  
Remove the following private functions from this chapter  
r.<Config_TAUXn_m>_disable_pek_interrupt  
r.<Config_TAUXn_m>_enable_pek_interrupt  
r.<Config_TAUXn_m>_disable_interrupt  
r.<Config_TAUXn_m>_enable_interrupt |
<table>
<thead>
<tr>
<th>Rev.</th>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1.20 | 3. API FUNCTIONS | **3.2.34 SCI3 Clock Synchronous Mode**  
Remove the following private functions from this chapter  
r.<Config_SCI3n>._enable_interrupt  
r.<Config_SCI3n>._disable_interrupt  

**3.2.35 SCI3 Asynchronous Mode**  
Remove the following private functions from this chapter  
r.<Config_SCI3n>._enable_interrupt  
r.<Config_SCI3n>._disable_interrupt  

**3.2.36 MSPI Master**  
Remove the following private functions from this chapter  
r.<Config_MSPIn_m>_enable_interrupt  
r.<Config_MSPIn_m>_disable_interrupt  

**3.2.37 MSPI Slave**  
Remove the following private functions from this chapter  
r.<Config_MSPIn_m>_enable_interrupt  
r.<Config_MSPIn_m>_disable_interrupt  

**3.2.39 Error Control Module**  
Remove the following private functions from this chapter  
r.<Config_ECM>_enable_pek_interrupt  
r.<Config_ECM>_disable_pek_interrupt  

**3.2.42 TIM Bit Compression Mode**  
Remove the following private functions from this chapter  
r.<Config_TIMn>_enable_interrupt  
r.<Config_TIMn>_disable_interrupt  

**3.2.43 TIM Gated Periodic Sampling Mode**  
Remove the following private functions from this chapter  
r.<Config_TIMn_m>_enable_interrupt  
r.<Config_TIMn_m>_disable_interrupt  

**3.2.44 TIM Input Prescaler Mode**  
Remove the following private functions from this chapter  
r.<Config_TIMn_m>_enable_interrupt  
r.<Config_TIMn_m>_disable_interrupt  

**3.2.45 TIM Input Event Mode**  
Remove the following private functions from this chapter  
r.<Config_TIMn_m>_enable_interrupt  
r.<Config_TIMn_m>_disable_interrupt  

**3.2.46 TIM Pulse Integration Mode**  
Remove the following private functions from this chapter  
r.<Config_TIMn_m>_enable_interrupt  
r.<Config_TIMn_m>_disable_interrupt |
<table>
<thead>
<tr>
<th>Rev.</th>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1.20 | 3. API FUNCTIONS | 3.2.47 TIM PWM Measurement Mode<br>Remove the following private functions from this chapter<br>`r_<Config_TIMn_m>_enable_interrupt`<br>`r_<Config_TIMn_m>_disable_interrupt`<br><br>3.2.48 TIM Serial Shift Mode<br>Remove the following private functions from this chapter<br>`r_<Config_TIMn_m>_enable_interrupt`<br>`r_<Config_TIMn_m>_disable_interrupt`<br><br>3.2.49 ATOM Signal Output Mode Compare<br>Remove the following private functions from this chapter<br>`r_<Config_ATOMn_m>_enable_interrupt`<br>`r_<Config_ATOMn_m>_disable_interrupt`<br><br>3.2.50 ATOM Signal Output Mode PWM<br>Remove the following private functions from this chapter<br>`r_<Config_ATOMn_m>_enable_interrupt`<br>`r_<Config_ATOMn_m>_disable_interrupt`<br><br>3.2.36 MSPI Master<br>Change the description of API[Syntax] part.<br>`R_<Config_MSPInm>_Send: Delete the redundant “const” in parameter in API [Syntax] part.`<br>`R_<Config_MSPInm>_Receive: Change the description of API`<br><br>3.2.39 Error Control Module<br>Change the API name in [Syntax] part.<br>`R_<Config_ECM>_Pseudo_XXXX_Start and R_<Config_ECM>_Pseudo_XXXX_Stop: Change the API name in [Syntax] part.`<br><br>Change "ADCJ0_Parity_Error" to "ADCJn_Parity_Error"
Add "Remark" in [Syntax] part.<br><br>Table 3.39.1 ECM pseudo error name:<br>Change the table name.<br>Table name: Change "ADCJ0_Parity_Error" to "ADCJn_Parity_Error"
Add "Remark" in [Syntax] part.<br><br>1.30 | 3. API FUNCTIONS | 3.2.3 CSI Slave:<br>`R_<Config_CSIXn>_Send: add “const” for parameter “tx_buf”`<br>`R_<Config_CSIXn>_Receive: add “const” for parameter “rx_buf”`<br><br>3.2.4 CSI Master:<br>Add function “R_<Config_CSIHn>_Send_Receive”<br>`R_<Config_CSIHn>_Send: add “const” for parameter “tx_buf”`<br>`R_<Config_CSIHn>_Receive: add “const” for parameter “rx_buf”`<br><br>3.2.4 CSI Master:<br>Add function “R_<Config_CSIHn>_Send_Receive”<br>`R_<Config_CSIXn>_Send: add “const” for parameter “tx_buf”`<br>`R_<Config_CSIXn>_Receive: add “const” for parameter “rx_buf”`<br><br>3.2.15 UART Interface<br>`R_<Config_UARTn>_Send, R_<Config_UARTn>_Receive: Add Remark3.`
<table>
<thead>
<tr>
<th>Rev.</th>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1.30 | 3. API FUNCTIONS | 3.2.34 SCI3 Clock Synchronous Mode  
\[ \text{R}_<\text{Config SCI3n}>\_\text{Send}: \text{adjust "const" position of parameter "tx_buf"} \]  
\[ \text{R}_<\text{Config SCI3n}>\_\text{Receive}: \text{add "const" for parameter "rx_buf"} \]  
3.2.35 SCI3 Asynchronous Mode  
\[ \text{R}_<\text{Config SCI3n}>\_\text{Send}: \text{adjust "const" position of parameter "tx_buf"} \]  
\[ \text{R}_<\text{Config SCI3n}>\_\text{Receive}: \text{add "const" for parameter "rx_buf"} \]  
\[ \text{R}_<\text{Config SCI3n}>\_\text{Multiprocessor Send}: \text{adjust "const" position of parameter "tx_buf"} \]  
\[ \text{R}_<\text{Config SCI3n}>\_\text{Multiprocessor Receive}: \text{add "const" for parameter "rx_buf"} \] |
| 1.40 | 2. OUTPUT FILES | Table 2.1 Output File List: Add the following files.  
\[ \text{r_cg_ad_air.h} \]  
\[ \text{r_cg_abfg.h} \]  
\[ \text{r_cg_ad_common.c} \]  
\[ \text{r_cg_ad_common.h} \] |
|      | 3. API FUNCTIONS | 3.2.1 General:  
The following functions are added:  
\[ \text{R\_ADC\_SyncStart}, \]  
\[ \text{R\_ADC\_TimerSyncStart}, \]  
\[ \text{R\_DMAC\_Create}, \]  
\[ \text{R\_DMA\_Suspend}, \]  
\[ \text{R\_DMA\_Resume}, \]  
\[ \text{R\_PDMAn\_Suspend}, \]  
\[ \text{R\_PDMAn\_Resume}, \]  
\[ \text{R\_DTS\_Create}, \]  
\[ \text{R\_DMA\_Start\_Transfer\_Error\_Interrupt}, \]  
\[ \text{R\_DMA\_Stop\_Transfer\_Error\_Interrupt}, \]  
\[ \text{r\_dmac\_error\_interrupt\_pe1}, \]  
\[ \text{R\_MSPI\_Master\_Create} \]  
Following function Remark description changed:  
\[ \text{R\_GTM\_Start\_Interrupt\_Error} \]  
\[ \text{R\_GTM\_Stop\_Interrupt\_Error} \]  
\[ \text{R\_ATOMn\_Start\_Interrupt\_IRQk} \]  
\[ \text{R\_ATOMn\_Stop\_interrupt\_IRQk} \] |
<table>
<thead>
<tr>
<th>Rev.</th>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.40</td>
<td>3. API FUNCTIONS</td>
<td>3.2.2 A/D Converter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The following functions are added:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R.&lt;Config_ADCXn&gt;_ScanGroupx_OperationOff,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R.&lt;Config_ADCXn&gt;_TH_Sampling_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R.&lt;Config_ADCXn&gt;_TH_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R.&lt;Config_ADCXn&gt;_VoltageDivider_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R.&lt;Config_ADCXn&gt;_VoltageDivider_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R.&lt;Config_ADCXn&gt;_StrongPullDown_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R.&lt;Config_ADCXn&gt;_StrongPullDown_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R.&lt;Config_ADCXn&gt;_SGDiag_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R.&lt;Config_ADCXn&gt;_SGDiag_OperationOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R.&lt;Config_ADCXn&gt;_SGDiag_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R.&lt;Config_ADCXn&gt;_SGDiag_GetResult</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R.&lt;Config_ADCXn&gt;_SGDiag_GetSubtractionResult</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R.&lt;Config_ADCXn&gt;_ASF_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R.&lt;Config_ADCXn&gt;_ASF_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R.&lt;Config_ADCXn&gt;_ASF_GetResult</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R.&lt;Config_ADCXn&gt;_ASF_GetRealTimeResult</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r.&lt;Config_ADCXn&gt;_sg_diag_end_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r.&lt;Config_ADCXn&gt;_mpx_request_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r.&lt;Config_ADCXn&gt;_asf_channelk_end_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Following functions are removed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R.&lt;Config_ADCXn&gt;_SyncStart</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R.&lt;Config_ADCXn&gt;_SyncTimerStart</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Following function name is updated:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Old:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R.&lt;Config_ADCXn&gt;_ScanGroupx_GetFloatingPointDataResult,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R.&lt;Config_ADCXn&gt;_ScanGroupx_GetFloatingPointResult,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Following function parameter is updated:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R.&lt;Config_ADCXn&gt;_ScanGroupx_GetFloatingPointResult,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R.&lt;Config_ADCXn&gt;_ScanGroupx_GetResult</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Following function description is updated:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R.&lt;Config_ADCXn&gt;_ScanGroupx_OperationOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R.&lt;Config_ADCXn&gt;_ScanGroupx_OperationOff</td>
</tr>
<tr>
<td></td>
<td>3.2.3 CSI Slave</td>
<td>R.&lt;Config_CSIxN&gt;_Send_Receive is added to support send and receive function</td>
</tr>
<tr>
<td></td>
<td>3.2.4 CSI Master</td>
<td>R.&lt;Config_CSIxN&gt;_Send_Receive is added to support send and receive function</td>
</tr>
<tr>
<td>Rev.</td>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>--------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1.40</td>
<td>3. API FUNCTIONS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.2.5 Interrupt</td>
<td>Following functions are added:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R._&lt;Config_INTC&gt;_IRQn_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R._&lt;Config_INTC&gt;_IRQn_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r._&lt;Config_INTC&gt;_intpn_interrupt_pek</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r._&lt;Config_INTC&gt;_irqn_interrupt</td>
</tr>
<tr>
<td></td>
<td>3.2.14 Stand-by Controller</td>
<td>Following functions are added:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R._&lt;Config_STBC&gt;_Prepare_Stop_Mode_Set_CPUCLK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R._&lt;Config_STBC&gt;_Prepare_Deep_Stop_Mode_Set_Clock_Mask</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R._&lt;Config_STBC&gt;_Prepare_Deep_Stop_Mode_Set_Clock_Source</td>
</tr>
<tr>
<td></td>
<td>3.2.16 Watchdog Timer</td>
<td>Following functions are added:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R._&lt;Config_WDTAn&gt;_Create</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R._&lt;Config_WDTAn&gt;_Restart</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R._&lt;Config_WDTAn&gt;_Create_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R._&lt;Config_WDTAn&gt;_Inerrupt</td>
</tr>
<tr>
<td></td>
<td>3.2.18 Data CRC</td>
<td>Following functions are added:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R._&lt;Config_DCRAn&gt;_InitializeCRCData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R._&lt;Config_KCRCn&gt;_InitializeCRCData</td>
</tr>
<tr>
<td></td>
<td>3.2.20 DMA Controller</td>
<td>Following functions are added:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r._&lt;Config_DMACnm&gt;_Callback_DMACnm_Transfer_Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R._&lt;Config_SDMACnm&gt;_Reset</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r._&lt;Config_SDMACnm&gt;_Callback_PEx_Address_Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Following function name is updated:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Old:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r._&lt;Config_DMACnm&gt;_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r._&lt;Config_DMACnm&gt;_dmacnm_interrupt</td>
</tr>
<tr>
<td></td>
<td>3.2.36 MSPI Master</td>
<td>Following function name is updated:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Old:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r._&lt;Config_MSPInm&gt;_callback_send</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r._&lt;Config_MSPInm&gt;_callback_receive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r._&lt;Config_MSPInm&gt;_interrupt_error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r._&lt;Config_MSPInm&gt;_callback_frameend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r._&lt;Config_MSPInm&gt;_Callback_Interrupt_Send</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r._&lt;Config_MSPInm&gt;_Callback_Interrupt_Receive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r._&lt;Config_MSPInm&gt;_Callback_Interrupt_Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r._&lt;Config_MSPInm&gt;_Callback_Interrupt_Frameend</td>
</tr>
<tr>
<td>Rev.</td>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1.40</td>
<td>3. API FUNCTIONS</td>
<td>3.2.39 Error Module Control</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Following functions are added:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R._&lt;Config_ECM&gt;_Set_Master_Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R._&lt;Config_ECM&gt;_Set_Checker_Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R._&lt;Config_ECM&gt;_Clear_Master_Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R._&lt;Config_ECM&gt;_Clear_Checker_Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R._&lt;Config_ECM&gt;_Delay_Timer_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R._&lt;Config_ECM&gt;_Delay_Timer_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r._&lt;Config_ECM&gt;_mi_interrupt_pek</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.2.42 TIM Bit Compression Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.2.43 TIM Gated Periodic Sampling Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.2.44 TIM Input Prescaler Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.2.45 TIM Input Event Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.2.46 TIM Pulse Integration Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.2.47 TIM PWM Measurement Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.2.48 TIM Serial Shift Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Following function name updated:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Old:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r._&lt;Config_TIMn&gt;_callback_gtm_error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r._&lt;Config_TIMn&gt;_Callback_GTM_Error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.2.49 ATOM Signal Output Mode Compare</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.2.51 ATOM Signal Output Mode PWM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.2.52 ATOM Signal Output Mode Serial</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Following function name is updated:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Old:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r._&lt;Config_ATOMn_m&gt;_callback_shared_interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r._&lt;Config_ATOMn_m&gt;_Callback_Shared_IRQk</td>
</tr>
<tr>
<td>Rev.</td>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>---------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| 1.40 | 3. API FUNCTIONS | 3.2.54 DTS Controller  
Following functions are removed:  
R.<Config_DTSn>_Enable_PEk_Interrupt  
R.<Config_DTSn>_Disable_PEk_Interrupt  
Following functions name is updated:  
Old:  
r.<Config_DTSn>_callback_dts_transfer_end_interrupt  
r.<Config_DTSn>_callback_dtsg_transfer_count_match_interrupt  
r.<Config_DTSn>_callback_pek_transfer_error_interrupt  
New:  
r.<Config_DTSn>._Callback_Dts_Transfer_End  
r.<Config_DTSn>._Callback_Dtsg_Transfer_Count_Match  
r.<Config_DTSn>._Callback_PEk_Transfer_Error  
Following function is added:  
r.<Config_DTSn>._Callback_Transfer_Error |
|       |         | 3.2.55 ADC Boundary Flag Generator  
New added component and relative API:  
R.<Config_ABFG>._Create  
R.<Config_ABFG>._Start  
R.<Config_ABFG>._Stop  
R.<Config_ABFG>._Create_UserInit  
r.<Config_ABFG>._boundary_flag_pulse_w_interrupt |
|       |         | Added usage example for 3.2.3 CSI Slave, 3.2.4 CSI Master, 3.2.14 Stand-by Controller,  
3.2.15 UART Interface, 3.2.32 IIC Master Mode, 3.2.33 IIC Slave Mode, 3.2.34 SCI3 Clock  
Synchronous Mode, 3.2.35 SCI3 Asynchronous Mode, 3.2.36 MSPI Master, 3.2.37 MSPI Slave. |
| 1.50 | 2. OUTPUT FILES | Table 2.4 Output File List:  
Delete old APIs from A/D Converter:  
R.<Config_ADCXn>._TH_Sampling_Stop  
Add new APIs to A/D Converter:  
R.<Config_ADCXn>._SelfDiagnostic_VoltageCircuitOn  
R.<Config_ADCXn>._SelfDiagnostic_VoltageCircuitOff  
R.<Config_ADCXn>._ScanGroupx_GetPWMDiagResult  
Add new APIs to One-Shot Pulse Output:  
R.<Config_TAUxn>._SoftwareTriggerOn  
Add a new output file to general part:  
r_smc_entry.h |
<table>
<thead>
<tr>
<th>Rev.</th>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1.50 | 3. API FUNCTIONS | 3.2.2 A/D Converter:  
Delete old APIs from A/D Converter:  
R_<Config_ADCXn>_TH_Sampling_Stop  
Add new APIs to A/D Converter in 3.2.2 A/D Converter  
R_<Config_ADCXn>_SelfDiagnostic_VoltageCircuitOn  
R_<Config_ADCXn>_SelfDiagnostic_VoltageCircuitOff  
R_<Config_ADCXn>_ScanGroupx_GetPWMDiagResult  
3.2.29 One-shot Pulse Output:  
Add new APIs to One-Shot Pulse Output:  
R_<Config_TAUXn>_SoftwareTriggerOn |

Smart Configurator User's Manual: RH850 API Reference

Publication Date:  
Rev.1.00 Jul.13.18  
Rev.1.50 Jul.20.23  
Published by:  
Renesas Electronics Corporation