User's Manual

Smart Configurator

User's Manual: RH850 API Reference

RENESAS MCU
RH850 Family

Renesas Electronics
www.renesas.com

Rev.1.60 Jan 2024

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 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 out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a 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 INVULNERABLE OR FREE FROM CORRUPTION, 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 resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
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 applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. 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.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(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/.

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. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.

2. Processing at power-on

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

3. Input of signal during power-off state

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

4. Handling of unused pins

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

5. Clock signals

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

6. Voltage application waveform at input pin

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

7. Prohibition of access to reserved addresses

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

8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a 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 FUNCITONS
How to Read This Manual	It is assumed that the readers of this manual have general knowledge of electricity, logic circuits, and microcontrollers.
Conventions	Deata significance: Higher digits on the left and lower digits on the right Active low representation: $\overline{\text{XXX}}$ (overscore over pin or signal name) Note: Footnote for item marked with Note in the text Caution: Information requiring particular attention Remark: Supplementary information Numeric representation: Decimal ... XXXX Hexadecimal ... 0xXXXX

TABLE OF CONTENTS

1. GENERAL

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 declaretion 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)

Peripheral Function	File Name	API Function Name
General	r_cg_main.c	main R_MAIN_UserInit
	r_cg_systeminit.c	R_Systeminit
	r_cg_cgc.c	R_CGC_Create
	r_cg_cgc_user.c	R_CGC_Create_UserInit
	r_cg_cgc.h	—
	r_smc_clock_info.h	—
	r_cg_intvector.c	—
	r_cg_intvector_PE*k*.c	—
	r_cg_intc_PE*k*.c	R_Interrupt_Initialize_ForPE
	r_cg_macrodriver.h	—
	r_cg_userdefine.h	—
	r_cg_ad.h	—
	r_cg_abfg.h	—
	r_cg_ad_common.c	R_ADC_SyncStart R_ADC_TimerSyncStart
	r_cg_ad_common.h	—
	r_cg_ad_air.h	—
	r_cg_csig.h	—
	r_cg_csih.h	—
	r_cg_dcra.h	—
	r_cg_dma.h	—

Table 2.2 Output File List (2/21)

Peripheral Function	File Name	API Function Name
General	r_cg_dma_common.c	R_DMAC_Create R_DMA_Suspend R_DMA_Resume R_PDMA*n*_Suspend R_PDMA*n*_Resume R_DMAC_Start_Error_Interrupt_PE*k* R_DMAC_Stop_Error_Interrupt_PE*k* R_DMA_Start_Transfer_Error_Interrupt R_DMA_Stop_Transfer_Error_Interrupt R_DTS_Create R_DTS_Suspend R_DTS_Resume R_DTS*g*_Start_Transfer_End_Interrupt R_DTS*g*_Stop_Transfer_End_Interrupt R_DTS*g*_Start_Transfer_Count_Match_Interrupt R_DTS*g*_Stop_Transfer_Count_Match_Interrupt
	r_cg_dma_common_user.c	r_sdmac_address_error_interrupt_pe*k* r_dts*g*_transfer_end_interrupt r_dts*g*_transfer_count_match_interrupt r_dmac_error_interrupt_pe*k*
	r_cg_dma_common.h	—
	r_cg_dtm.h	—
	r_cg_dts.h	—
	r_cg_dts_common.c	R_DTS_Suspend R_DTS_Resume R_DTS*g*_Start_Transfer_End_Interrupt R_DTS*g*_Stop_Transfer_End_Interrupt R_DTS*g*_Start_Transfer_Count_Match_Interrupt R_DTS*g*_Stop_Transfer_Count_Match_Interrupt R_DTS_Start_PE*k*_Transfer_Error_Interrupt R_DTS_Stop_PE*k*_Transfer_Error_Interrupt
	r_cg_dts_common_user.c	r_dts*g*_transfer_end_interrupt r_dts*g*_transfer_count_match_interrupt r_dts_transfer_error_interrupt_pe*k*
	r_cg_dts_common.h	—
	r_cg_ecm.h	—
	r_cg_gtm_clock.h	—
	r_cg_gtm_common.h	—
	r_cg_gtm_common.c	R_GTM_Start_Interrupt_Error R_GTM_Stop_Interrupt_Error
	r_cg_gtm_common_user.c	r_gtm_error_interrupt

Table 2.3 Output File List (3/21)

Peripheral Function	File Name	API Function Name
General	r_cg_atom.h	—
	r_cg_atom_common.c	R_ATOM*n*_Start_Interrupt_IRQ*k* R_ATOM*n*_Stop_Interrupt_IRQ*k*
	r_cg_atom_common_user.c	r_atom*n*_irq*k*_interrupt
	r_cg_intc.h	—
	r_cg_kcrc.h	—
	r_cg_key.h	—
	r_cg_mspi.h	—
	r_cg_mspi_common.c	R_MSPI_Master_Create R_MSPI_Start_Interrupt_MSPI*n*TX R_MSPI_Stop_Interrupt_MSPI*n*TX R_MSPI_Start_Interrupt_MSPI*n*RX R_MSPI_Stop_Interrupt_MSPI*n*RX R_MSPI_Start_Interrupt_MSPI*n*FE R_MSPI_Stop_Interrupt_MSPI*n*FE R_MSPI_Start_Interrupt_MSPI*n*ERR R_MSPI_Stop_Interrupt_MSPI*n*ERR
	r_cg_mspi_common_user.c	r_mspi*n*_interrupt_send r_mspi*n*_interrupt_receive r_mspi*n*_interrupt_frameend r_mspi*n*_interrupt_error
	r_cg_mspi_common.h	
	r_cg_ostm.h	—
	r_cg_port.h	—
	r_cg_riic.h	—
	r_cg_rtc.h	—
	r_cg_sci3.h	—
	r_cg_stbc.h	—
	r_cg_taub.h	—
	r_cg_taud.h	—
	r_cg_tauj.h	—
	r_cg_tbu.h	—
	r_cg_tim.h	—
	r_cg_uart.h	—
	r_cg_wdt.h	—
	r_smc_interrupt.c	R_Interrupt_Create
	r_smc_interrupt.h	—
	r_smc_intprg.c	—
	r_smc_entry.h	—
	Pin.c	R_Pins_Create
	Pin.h	—

Table 2.4 Output File List (4/21)

Peripheral Function	File Name	API Function Name
A/D Converter	<Config_ADC*Xn*>.c	R_<Config_ADC*Xn*>_Create R_<Config_ADCX*n*>_Halt R_<Config_ADCXn>_SelfDiagnostic_VoltageCircuitOn R_<Config_ADCXn>_SelfDiagnostic_VoltageCircuitOff R_<Config_ADC*Xn*>_ScanGroup*x*_Start R_<Config_ADC*Xn*>_ScanGroup*x*_OperationOn R_<Config_ADC*Xn*>_ScanGroup*x*_OperationOff R_<Config_ADC*Xn*>_ScanGroup*x*_GetResult R_<Config_ADCXn>_ScanGroupx_GetPWMDiagResult R_<Config_ADC*Xn*>_ScanGroup*x*_GetFloatingPointResult R_<Config_ADC*Xn*>_SetMultiplexerCommand R_<Config_ADC*Xn*>_TH_Group*k*_Start R_<Config_ADC*Xn*>_TH_Sampling_Start R_<Config_ADC*Xn*>_TH_Stop R_<Config_ADC*Xn*>_ScanGroup*x*_TimerStart R_<Config_ADC*Xn*>_ScanGroup*x*_TimerStop R_<Config_ADC*Xn*>_VoltageDivider_Start R_<Config_ADC*Xn*>_VoltageDivider_Stop R_<Config_ADC*Xn*>_StrongPullDown_Start R_<Config_ADC*Xn*>_StrongPullDown_Stop R_<Config_ADC*Xn*>_SGDiag_Start R_<Config_ADC*Xn*>_SGDiag_OperationOn R_<Config_ADC*Xn*>_SGDiag_OperationOff R_<Config_ADC*Xn*>_SGDiag_GetResult R_<Config_ADC*Xn*>_SGDiag_GetSubtractionResult R_<Config_ADC*Xn*>_ASF_Start R_<Config_ADC*Xn*>_ASF_Stop R_<Config_ADC*Xn*>_ASF_GetResult R_<Config_ADC*Xn*>_ASF_GetRealTimeResult
	<Config_ADC*Xn*>_user.c	R_<Config_ADCX*n*>_Create_UserInit r_<Config_ADCX*n*>_error_interrupt r_<Config_ADCX*n*>_scan_group*x*_end_interrupt r_<Config_ADCX*n*>_sg_diag_end_interrupt r_<Config_ADCX*n*>_mpx_request_interrupt r_<Config_ADCX*n*>_asf_channel*k*_end_interrupt r_<Config_ADCX*n*>_monitor_error_interrupt
	<Config_ADC*Xn*>.h	—

Table 2.5 Output File List (5/21)

Peripheral Function	File Name	API Function Name
CSI Slave	<Config_CSI*Xn*>.c	R_<Config_CSI*Xn*>_Create R_<Config_CSI*Xn*>_Start R_<Config_CSI*Xn*>_Stop R_<Config_CSI*Xn*>_Send R_<Config_CSI*Xn*>_Receive R_<Config_CSI*Xn*>_Send_Receive
	<Config_CSI*Xn*>_user.c	R_<Config_CSI*Xn*>_Create_UserInit r_<Config_CSI*Xn*>_interrupt_send r_<Config_CSI*Xn*>_interrupt_receive r_<Config_CSI*Xn*>_interrupt_error
	<Config_CSI*Xn*>.h	–
CSI Master	<Config_CSI*Xn*>.c	R_<Config_CSI*Xn*>_Create R_<Config_CSI*Xn*>_Start R_<Config_CSI*Xn*>_Stop R_<Config_CSI*Xn*>_Send R_<Config_CSI*Xn*>_Receive R_<Config_CSI*Xn*>_Send_Receive
	<Config_CSI*Xn*>_user.c	R_<Config_CSI*Xn*>_Create_UserInit r_<Config_CSI*Xn*>_interrupt_send r_<Config_CSI*Xn*>_interrupt_receive r_<Config_CSI*Xn*>_interrupt_error
	<Config_CSI*Xn*>.h	–
Interrupt	<Config_INTC>.c	R_<Config_INTC>_Create R_<Config_INTC>_INTP*n*_Start R_<Config_INTC>_INTP*n*_Stop R_<Config_INTC>_IRQn_Start R_<Config_INTC>_IRQn_Stop R_<Config_INTC>_NMI_Start R_<Config_INTC>_NMI_Stop R_<Config_INTC>_SINT*n*_Start R_<Config_INTC>_SINT*n*_Stop R_<Config_INTC>_SINT*n*_TriggerOn
	<Config_INTC>_user.c	R_<Config_INTC>_Create_UserInit r_<Config_INTC>_intp*n*_interrupt r_<Config_INTC>_intp*n*_interrupt_pe*k* r_<Config_INTC>_irq*n*_interrupt r_<Config_INTC>_nmi_interrupt r_<Config_INTC>_sint1_interrupt_pe*k*
	<Config_INTC>.h	–

Table 2.6 Output File List (6/21)

Peripheral Function	File Name	API Function Name
Input Interval Timer	<Config_TAU*Xn_m*>.c	R_<Config_TAU*Xn_m*>_Create R_<Config_TAU*Xn_m*>_Start R_<Config_TAU*Xn_m*>_Stop
	<Config_TAU*Xn_m*>_user.c	R_<Config_TAU*Xn_m*>_Create_UserInit r_<Config_TAU*Xn_m*>_interrupt r_<Config_TAU*Xn_m*>_interrupt_pe*k*
	<Config_TAU*Xn_m*>.h	–
Input Pulse Interval Measurement	<Config_TAU*Xn_m*>.c	R_<Config_TAU*Xn_m*>_Create R_<Config_TAU*Xn_m*>_Start R_<Config_TAU*Xn_m*>_Stop R_<Config_TAU*Xn_m*>_Get_PulseWidth
	<Config_TAU*Xn_m*>_user.c	R_<Config_TAU*Xn_m*>_Create_UserInit r_<Config_TAU*Xn_m*>_interrupt r_<Config_TAU*Xn_m*>_interrupt_pe*k*
	<Config_TAU*Xn_m*>.h	–
Interval Timer	<Config_TAU*Xn_m*>.c	R_<Config_TAU*Xn_m*>_Create R_<Config_TAU*Xn_m*>_Start R_<Config_TAU*Xn_m*>_Stop
	<Config_TAU*Xn_m*>_user.c	R_<Config_TAU*Xn_m*>_Create_UserInit r_<Config_TAU*Xn_m*>_interrupt r_<Config_TAU*Xn_m*>_interrupt_pe*k*
	<Config_TAU*Xn_m*>.h	–

Table 2.7 Output File List (7/21)

Peripheral Function	File Name	API Function Name
Triangle PWM Output	<Config_TAU*Xn*>.c	R_<Config_TAU*Xn*>_Create R_<Config_TAU*Xn*>_Start R_<Config_TAU*Xn*>_Stop
	<Config_TAU*Xn*>_user.c	R_<Config_TAU*Xn*>_Create_UserInit r_<Config_TAU*Xn*>_channel*m*_interrupt r_<Config_TAU*Xn*>_channel*m*_interrupt_pe*k*
	<Config_TAU*Xn*>.h	–
Triangle PWM Output With Deadtime	<Config_TAU*Xn*>.c	R_<Config_TAU*Xn*>_Create R_<Config_TAU*Xn*>_Start R_<Config_TAU*Xn*>_Stop
	<Config_TAU*Xn*>_user.c	R_<Config_TAU*Xn*>_Create_UserInit r_<Config_TAU*Xn*>_channel*m*_interrupt r_<Config_TAU*Xn*>_channel*m*_interrupt_pe*k*
	<Config_TAU*Xn*>.h	–
OS Timer	<Config_OSTM*n*>.c	R_<Config_OSTM*n*>_Create R_<Config_OSTM*n*>_Start R_<Config_OSTM*n*>_Stop R_<Config_OSTM*n*>_Set_CompareValue
	<Config_OSTM*n*>_user.c	R_<Config_OSTM*n*>_Create_UserInit r_<Config_OSTM*n*>_interrupt
	<Config_OSTM*n*>.h	–
Port	<Config_PORT>.c	R_<Config_PORT>_Create
	<Config_PORT>_user.c	R_<Config_PORT>_Create_UserInit
	<Config_PORT>.h	–
PWM Output	<Config_TAU*Xn*>.c	R_<Config_TAU*Xn*>_Create R_<Config_TAU*Xn*>_Start R_<Config_TAU*Xn*>_Stop
	<Config_TAU*Xn*>_user.c	R_<Config_TAU*Xn*>_Create_UserInit r_<Config_TAU*Xn*>_channel*m*_interrupt r_<Config_TAU*Xn*>_channel*m*_interrupt_pe*k*
	<Config_TAU*Xn*>.h	–

Table 2.8 Output File List (8/21)

Peripheral Function	File Name	API Function Name
Stand-by Controller	<Config_STBC>.c	R_<Config_STBC>_Prepare_Stop_Mode R_<Config_STBC>_Start_Stop_Mode R_<Config_STBC>_Prepare_Deep_Stop_Mode R_<Config_STBC>_Start_Deep_Stop_Mode R_<Config_STBC>_Deep_Stop_Loop R_<Config_STBC>_Set_Module_*XXXX*_Standby_Mode R_<Config_STBC>_Cancel_Module_*XXXX*_Standby_Mode
	<Config_STBC>_user.c	R_<Config_STBC>_Prepare_Stop_Mode_Set_Peripheral R_<Config_STBC>_Prepare_Stop_Mode_Set_Interrupt R_<Config_STBC>_Prepare_Stop_Mode_Set_Clock_Mask R_<Config_STBC>_Prepare_Stop_Mode_Set_Clock_Source R_<Config_STBC>_Prepare_Stop_Mode_Set_CPUCLK R_<Config_STBC>_Prepare_Deep_Stop_Mode_Set_Peripheral R_<Config_STBC>_Prepare_Deep_Stop_Mode_Set_Interrupt R_<Config_STBC>_Prepare_Deep_Stop_Mode_Set_Clock_Mask R_<Config_STBC>_Prepare_Deep_Stop_Mode_Set_Clock_Source R_<Config_STBC>_Check_Module_*XXXX*_Idle_State
	<Config_STBC>.h	—
UART Interface	<Config_UART*n*>.c	R_<Config_UART*n*>_Create R_<Config_UART*n*>_Start R_<Config_UART*n*>_Stop R_<Config_UART*n*>_Send R_<Config_UART*n*>_Receive
	<Config_UART*n*>_user.c	R_<Config_UART*n*>_Create_UserInit r_<Config_UART*n*>_interrupt_send r_<Config_UART*n*>_interrupt_receive r_<Config_UART*n*>_interrupt_error
	<Config_UART*n*>.h	—
Watchdog Timer	<Config_WDT*n*>.c <Config_WDTX*n*>.c	R_<Config_WDT*n*>_Create R_<Config_WDT*n*>_Restart R_<Config_WDTX*n*>_Create R_<Config_WDTX*n*>_Restart
	<Config_WDT*n*>_user.c <Config_WDTX*n*>_user.c	R_<Config_WDT*n*>_Create_UserInit r_<Config_WDT*n*>_interrupt R_<Config_WDTX*n*>_Create_UserInit r_<Config_WDTX*n*>_interrupt
	<Config_WDT*n*>.h	—

Table 2.9 Output File List (9/21)

Peripheral Function	File Name	API Function Name
Clock Divide	<Config_TAU*Xn*>.c	R_<Config_TAU*Xn*>_Create R_<Config_TAU*Xn_m*>_Start R_<Config_TAU*Xn_m*>_Stop
	<Config_TAUXn>_user.c	R_<Config_TAU*Xn*>_Create_UserInit r_<Config_TAU*Xn_m*>_interrupt r_<Config_TAU*Xn_m*>_interrupt_pe*k*
	<Config_TAU*Xn*>.h	—
Data CRC	<Config_DCRA*n*>.c <Config_KCRC*n*>.c	R_<Config_DCRA*n*>_Create R_<Config_DCRAn>_InitializeCRCData R_<Config_DCRA*n*>_Input32bitData R_<Config_DCRA*n*>_Input16bitData R_<Config_DCRA*n*>_Input8bitData R_<Config_DCRA*n*>_GetResult_32bitData R_<Config_DCRA*n*>_GetResult_16bitData R_<Config_KCRC*n*>_Create R_<Config_KCRCn>_InitializeCRCData R_<Config_KCRC*n*>_Input32bitData R_<Config_KCRC*n*>_Input16bitData R_<Config_KCRC*n*>_Input8bitData R_<Config_KCRC*n*>_GetResult_64bitData R_<Config_KCRC*n*>_GetResult_32bitData R_<Config_KCRC*n*>_GetResult_16bitData R_<Config_KCRC*n*>_GetResult_8bitData
	<Config_DCRA*n*>_user.c	R_<Config_DCRA*n*>_Create_UserInit R_<Config_KCRC*n*>_Create_UserInit
	<Config_DCRA*n*>.h	—
Delay Count	<Config_TAU*Xn_m*>.c	R_<Config_TAU*Xn_m*>_Create R_<Config_TAU*Xn_m*>_Start R_<Config_TAU*Xn_m*>_Stop
	<Config_TAU*Xn_m*>_user.c	R_<Config_TAU*Xn_m*>_Create_UserInit r_<Config_TAU*Xn_m*>_interrupt r_<Config_TAU*Xn_m*>_interrupt_pe*k*
	<Config_TAU*Xn_m*>.h	—

Table 2.10 Output File List (10/21)

Peripheral Function	File Name	API Function Name
DMA Controller	<Config_DMAC*n*>.c	R_<Config_DMAC*n*>_Create R_<Config_DMAC*nm*>_Start R_<Config_DMAC*nm*>_Stop R_<Config_DMAC*nm*>_Set_SoftwareTrigger R_<Config_DMAC*nm*>_Suspend R_<Config_DMAC*nm*>_Resume R_<Config_SDMAC*nm*>_Create R_<Config_SDMAC*nm*>_Start R_<Config_SDMAC*nm*>_Stop R_<Config_SDMAC*nm*>_Resume R_<Config_SDMACnm>_Reset R_<Config_SDMAC*nm*>_Set_DescriptorMemory
	<Config_DMAC*n*>_user.c	R_<Config_DMAC*nm*>_Create_UserInit r_<Config_DMAC*nm*>_dmac*nm*_interrupt r_<Config_DMACnm>_Callback_DMAnm_Transfer_Error R_<Config_SDMAC*nm*>_Create_UserInit r_<Config_SDMAC*nm*>_end_interrupt r_<Config_SDMACnm>_Callback_PEx_Address_Error
	<Config_DMAC*n*>.h	—
External Event Count	<Config_TAU*Xn_m*>.c	R_<Config_TAU*Xn_m*>_Create R_<Config_TAU*Xn_m*>_Start R_<Config_TAU*Xn_m*>_Stop
	<Config_TAU*Xn_m*>_user.c	R_<Config_TAU*Xn_m*>_Create_UserInit r_<Config_TAU*Xn_m*>_interrupt r_<Config_TAU*Xn_m*>_interrupt_pe*k*
	<Config_TAU*Xn_m*>.h	—
Input Period Count Detection	<Config_TAU*Xn_m*>.c	R_<Config_TAU*Xn_m*>_Create R_<Config_TAU*Xn_m*>_Start R_<Config_TAU*Xn_m*>_Stop R_<Config_TAU*Xn_m*>_Get_PulseWidth
	<Config_TAU*Xn_m*>_user.c	R_<Config_TAU*Xn_m*>_Create_UserInit r_<Config_TAU*Xn_m*>_interrupt r_<Config_TAU*Xn_m*>_interrupt_pe*k*
	<Config_TAU*Xn_m*>.h	—

Table 2.11 Output File List (11/21)

Peripheral Function	File Name	API Function Name
Input Position Detection Function	<Config_TAU*Xn_m*>.c	R_<Config_TAU*Xn_m*>_Create R_<Config_TAU*Xn_m*>_Start R_<Config_TAU*Xn_m*>_Stop R_<Config_TAU*Xn_m*>_Get_PulseWidth
	<Config_TAU*Xn_m*>_user.c	R_<Config_TAU*Xn_m*>_Create_UserInit r_<Config_TAU*Xn_m*>_interrupt r_<Config_TAU*Xn_m*>_interrupt_pe*k*
	<Config_TAU*Xn_m*>.h	—
Input Pulse Interval Judgment	<Config_TAU*Xn_m*>.c	R_<Config_TAU*Xn_m*>_Create R_<Config_TAU*Xn_m*>_Start R_<Config_TAU*Xn_m*>_Stop
	<Config_TAU*Xn_m*>_user.c	R_<Config_TAU*Xn_m*>_Create_UserInit r_<Config_TAU*Xn_m*>_interrupt r_<Config_TAU*Xn_m*>_interrupt_pe*k*
	<Config_TAU*Xn_m*>.h	—
Input Signal Width Judgment	<Config_TAU*Xn_m*>.c	R_<Config_TAU*Xn_m*>_Create R_<Config_TAU*Xn_m*>_Start R_<Config_TAU*Xn_m*>_Stop
	<Config_TAU*Xn_m*>_user.c	R_<Config_TAU*Xn_m*>_Create_UserInit r_<Config_TAU*Xn_m*>_interrupt r_<Config_TAU*Xn_m*>_interrupt_pe*k*
	<Config_TAU*Xn_m*>.h	—
Input Signal Width Measurement.	<Config_TAU*Xn_m*>.c	R_<Config_TAU*Xn_m*>_Create R_<Config_TAU*Xn_m*>_Start R_<Config_TAU*Xn_m*>_Stop R_<Config_TAU*Xn_m*>_Get_PulseWidth
	<Config_TAU*Xn_m*>_user.c	R_<Config_TAU*Xn_m*>_Create_UserInit r_<Config_TAU*Xn_m*>_interrupt r_<Config_TAU*Xn_m*>_interrupt_pe*k*
	<Config_TAU*Xn_m*>.h	—
Key Interrupt Function	<Config_KEY>.c	R_<Config_KEY>_Create R_<Config_KEY>_Start R_<Config_KEY>_Stop
	<Config_KEY>_user.c	R_<Config_KEY>_Create_UserInit r_<Config_KEY>_interrupt
	<Config_KEY>.h	—

Table 2.12 Output File List (12/21)

Peripheral Function	File Name	API Function Name
One Pulse Output	<Config_TAU*Xn_m*>.c	R_<Config_TAU*Xn_m*>_Create R_<Config_TAU*Xn_m*>_Start R_<Config_TAU*Xn_m*>_Stop
	<Config_TAU*Xn_m*>_user.c	R_<Config_TAU*Xn_m*>_Create_UserInit r_<Config_TAU*Xn_m*>_interrupt r_<Config_TAU*Xn_m*>_interrupt_pe*k*
	<Config_TAU*Xn_m*>.h	–
One-Shot Pulse Output	<Config_TAU*Xn*>.c	R_<Config_TAU*Xn*>_Create R_<Config_TAU*Xn*>_Start R_<Config_TAU*Xn*>_Stop R_<Config_TAU*Xn*>_SoftwareTriggerOn
	<Config_TAU*Xn*>_user.c	R_<Config_TAU*Xn*>_Create_UserInit r_<Config_TAU*Xn*>_channel*m*_interrupt r_<Config_TAU*Xn*>_channel*m*_interrupt_pe*k*
	<Config_TAU*Xn*>.h	–
Overflow Interrupt Output (Input period count detecting)	<Config_TAU*Xn_m*>.c	R_<Config_TAU*Xn_m*>_Create R_<Config_TAU*Xn_m*>_Start R_<Config_TAU*Xn_m*>_Stop
	<Config_TAU*Xn_m*>_user.c	R_<Config_TAU*Xn_m*>_Create_UserInit r_<Config_TAU*Xn_m*>_interrupt r_<Config_TAU*Xn_m*>_interrupt_pe*k*
	<Config_TAU*Xn_m*>.h	–
Overflow Interrupt Output (Width Measurement)	<Config_TAU*Xn_m*>.c	R_<Config_TAU*Xn_m*>_Create R_<Config_TAU*Xn_m*>_Start R_<Config_TAU*Xn_m*>_Stop
	<Config_TAU*Xn_m*>_user.c	R_<Config_TAU*Xn_m*>_Create_UserInit r_<Config_TAU*Xn_m*>_interrupt r_<Config_TAU*Xn_m*>_interrupt_pe*k*
	<Config_TAU*Xn_m*>.h	–

Table 2.13 Output File List (13/21)

Peripheral Function	File Name	API Function Name
IIC Master Mode	<Config_RIIC*n*>.c	R_<Config_RIIC*n*>_Create R_<Config_RIIC*n*>_Start R_<Config_RIIC*n*>_Stop R_<Config_RIIC*n*>_Master_Send R_<Config_RIIC*n*>_Master_Send_Without_Stop R_<Config_RIIC*n*>_Master_Receive R_<Config_RIIC*n*>_StartCondition R_<Config_RIIC*n*>_StopCondition
	<Config_RIIC*n*>_user.c	R_<Config_RIIC*n*>_Create_UserInit r_<Config_RIIC*n*>_error_interrupt r_<Config_RIIC*n*>_receive_interrupt r_<Config_RIIC*n*>_transmit_interrupt r_<Config_RIIC*n*>_transmitend_interrupt r_<Config_RIIC*n*>_callback_transmitend r_<Config_RIIC*n*>_callback_receiveend r_<Config_RIIC*n*>_callback_receiveerror
	<Config_RIIC*n*>.h	–
IIC Slave Mode	<Config_RIIC*n*>.c	R_<Config_RIIC*n*>_Create R_<Config_RIIC*n*>_Start R_<Config_RIIC*n*>_Stop R_<Config_RIIC*n*>_Slave_Send R_<Config_RIIC*n*>_Slave_Receive R_<Config_RIIC*n*>_StartCondition R_<Config_RIIC*n*>_StopCondition
	<Config_RIIC*n*>_user.c	R_<Config_RIIC*n*>_Create_UserInit r_<Config_RIIC*n*>_error_interrupt r_<Config_RIIC*n*>_receive_interrupt r_<Config_RIIC*n*>_transmit_interrupt r_<Config_RIIC*n*>_transmitend_interrupt r_<Config_RIIC*n*>_callback_transmitend r_<Config_RIIC*n*>_callback_receiveend r_<Config_RIIC*n*>_callback_receiveerror
	<Config_RIIC*n*>.h	–

Table 2.14 Output File List (14/21)

Peripheral Function	File Name	API Function Name
SCI3 Clock Synchronous Mode	<Config_SCI3*n*>.c	R_<Config_SCI3*n*>_Create R_<Config_SCI3*n*>_Start R_<Config_SCI3*n*>_Stop R_<Config_SCI3*n*>_Send R_<Config_SCI3*n*>_Receive
	<Config_SCI3*n*>_user.c	R_<Config_SCI3*n*>_Create_UserInit r_<Config_SCI3*n*>_interrupt_send r_<Config_SCI3*n*>_interrupt_receive r_<Config_SCI3*n*>_interrupt_error r_<Config_SCI3*n*>_interrupt_sendend r_<Config_SCI3*n*>_callback_sendend r_<Config_SCI3*n*>_callback_receiveend r_<Config_SCI3*n*>_callback_error
	<Config_SCI3*n*>.h	–
SCI3 Asynchronous Mode	<Config_SCI3*n*>.c	R_<Config_SCI3*n*>_Create R_<Config_SCI3*n*>_Start R_<Config_SCI3*n*>_Stop R_<Config_SCI3*n*>_Send R_<Config_SCI3*n*>_Receive R_<Config_SCI3*n*>_Multiprocessor_Send R_<Config_SCI3*n*>_Multiprocessor_Receive
	<Config_SCI3*n*>_user.c	R_<Config_SCI3*n*>_Create_UserInit r_<Config_SCI3*n*>_interrupt_send r_<Config_SCI3*n*>_interrupt_receive r_<Config_SCI3*n*>_interrupt_error r_<Config_SCI3*n*>_interrupt_sendend r_<Config_SCI3*n*>_callback_sendend r_<Config_SCI3*n*>_callback_receiveend r_<Config_SCI3*n*>_callback_error
	<Config_SCI3*n*>.h	–

Table 2.15 Output File List (15/21)

Peripheral Function	File Name	API Function Name
MSPI Master	<Config_MSPI*nm*>.c	R_<Config_MSPI*nm*>_Create R_<Config_MSPI*nm*>_Start R_<Config_MSPI*nm*>_Stop R_<Config_MSPI*nm*>_Send R_<Config_MSPI*nm*>_Receive R_<Config_MSPI*nm*>_Software_Trigger
	<Config_MSPI*nm*>_user.c	R_<Config_MSPI*nm*>_Create_UserInit r_<Config_MSPI*nm*>_channel*nm*_interrupt_send r_<Config_MSPI*nm*>_channel*nm*_interrupt_receive r_<Config_MSPI*nm*>_Callback_Interrupt_Send r_<Config_MSPI*nm*>_Callback_Interrupt_Receive r_<Config_MSPI*nm*>_Callback_Interrupt_Error r_<Config_MSPI*nm*>_Callback_Interrupt_Frameend r_<Config_MSPI*nm*>_callback_sendend r_<Config_MSPI*nm*>_callback_receiveend r_<Config_MSPI*nm*>_callback_error
	<Config_MSPI*nm*>.h	–
MSPI Slave	<Config_MSPI*n*>.c	R_<Config_MSPI*n*>_Create R_<Config_MSPI*n*>_Start R_<Config_MSPI*n*>_Stop R_<Config_MSPI*n*>_Send R_<Config_MSPI*n*>_Receive R_<Config_MSPI*n*>_Software_Trigger
	<Config_MSPI*n*>_user.c	R_<Config_MSPI*n*>_Create_UserInit r_<Config_MSPI*n*>_interrupt_send r_<Config_MSPI*n*>_interrupt_receive r_<Config_MSPI*n*>_interrupt_frameend r_<Config_MSPI*n*>_interrupt_error r_<Config_MSPI*n*>_channel*n*0_interrupt_send r_<Config_MSPI*n*>_channel*n*0_interrupt_receive r_<Config_MSPI*n*>_callback_receiveend r_<Config_MSPI*n*>_callback_sendend r_<Config_MSPI*n*>_callback_error
	<Config_MSPI*n*>.h	–

Table 2.16 Output File List (16/21)

Peripheral Function	File Name	API Function Name
Real-Time Clock	<Config_RTC>.c	R_<Config_RTC*n*>_Create R_<Config_RTC*n*>_Start R_<Config_RTC*n*>_Stop R_<Config_RTC*n*>_Set_HourSystem R_<Config_RTC*n*>_Get_CounterBufferValue R_<Config_RTC*n*>_Get_CounterDirectValue R_<Config_RTC*n*>_Set_CounterValue R_<Config_RTC*n*>_Get_SubCounterValue R_<Config_RTC*n*>_Set_ErrorCorrectionValue R_<Config_RTC*n*>_Set_SubCounterCompareValue R_<Config_RTC*n*>_Get_AlarmValue R_<Config_RTC*n*>_Set_AlarmValue R_<Config_RTC*n*>_Set_AlarmOn R_<Config_RTC*n*>_Set_AlarmOff R_<Config_RTC*n*>_Set_ConstPeriodInterruptOn R_<Config_RTC*n*>_Set_ConstPeriodInterruptOff R_<Config_RTC*n*>_Set_1secondInterruptOn R_<Config_RTC*n*>_Set_1secondInterruptOff R_<Config_RTC*n*>_Set_RTCA1HZOn R_<Config_RTC*n*>_Set_RTCA1HZOff
	<Config_RTC>_user.c	R_<Config_RTC*n*>_Create_UserInit R_<Config_RTC*n*>_interrupt_alarm R_<Config_RTC*n*>_interrupt_periodic R_<Config_RTC*n*>_interrupt_1second
	<Config_RTC>.h	—
Error Control Module	<Config_ECM>.c	R_<Config_ECM>_Create R_<Config_ECM>_Start R_<Config_ECM>_Stop R_<Config_ECM>_Set_Master_Error R_<Config_ECM>_Set_Checker_Error R_<Config_ECM>_Clear_Master_Error R_<Config_ECM>_Clear_Checker_Error R_<Config_ECM>_Delay_Timer_Start R_<Config_ECM>_Delay_Timer_Stop R_<Config_ECM>_Pseudo_XXXX_Start R_<Config_ECM>_Pseudo_XXXX_Stop
	<Config_ECM>_user.c	R_<Config_ECM>_Create_UserInit r_<Config_ECM>_ecmmi_interrupt_pek r_<Config_ECM>_mi_interrupt_pek r_<Config_ECM>_dclsmi_interrupt_pek
	<Config_ECM>.h	—

Table 2.17 Output File List (17/21)

Peripheral Function	File Name	API Function Name
GTM Clock	<Config_GTM_Clock>.c	R_<Config_GTM_Clock>_Create R_GTM_CCM*n*_Create
	<Config_GTM_Clock>_user.c	R_<Config_GTM_Clock>_Create_UserInit
	<Config_GTM_Clock>.h	—
Time Base Unit	<Config_TBU>.c	R_<Config_TBU>_Create R_<Config_TBU>_Start R_<Config_TBU>_Stop
	<Config_TBU>_user.c	R_<Config_TBU>_Create_UserInit
	<Config_TBU>.h	—
TIM Bit Compression Mode	<Config_TIM*n*>.c	R_<Config_TIM*n*>_Create R_<Config_TIM*n*>_Start R_<Config_TIM*n*>_Stop R_<Config_TIM*n*>_SetDetectedEdge R_<Config_TIM*n*>_Software_Reset R_<Config_TIM*n*>_NEWVAL_TriggerOn R_<Config_TIM*n*>_ECNTOFL_TriggerOn R_<Config_TIM*n*>_CNTOFL_TriggerOn R_<Config_TIM*n*>_GPROFL_TriggerOn R_<Config_TIM*n*>_TODETOFL_TriggerOn R_<Config_TIM*n*>_Channel*m*_GLITCHDETOFL_TriggerOn
	<Config_TIM*n*>_user.c	R_<Config_TIM*n*>_Create_UserInit r_<Config_TIM*n*>_share_interrupt r_<Config_TIM*n*>_ch*m*_share_interrupt r_<Config_TIM*n*>_Callback_GTM_Error
	<Config_TIM*n*>.h	—
TIM Gated Periodic Sampling Mode	<Config_TIM*n*_*m*>.c	R_<Config_TIM*n*_*m*>_Create R_<Config_TIM*n*_*m*>_Start R_<Config_TIM*n*_*m*>_Stop R_<Config_TIM*n*_*m*>_UpdateElapsedClock R_<Config_TIM*n*_*m*>_Software_Reset R_<Config_TIM*n*_*m*>_NEWVAL_TriggerOn R_<Config_TIM*n*_*m*>_ECNTOFL_TriggerOn R_<Config_TIM*n*_*m*>_CNTOFL_TriggerOn R_<Config_TIM*n*_*m*>_GPROFL_TriggerOn R_<Config_TIM*n*_*m*>_TODETOFL_TriggerOn R_<Config_TIM*n*_*m*>_GLITCHDETOFL_TriggerOn
	<Config_TIM*n*_*m*>_user.c	R_<Config_TIM*n*_*m*>_Create_UserInit r_<Config_TIM*n*_*m*>_Callback_GTM_Error r_<Config_TIM*n*_*m*>_share_interrupt
	<Config_TIM*n*_*m*>.h	—

Table 2.18 Output File List (18/21)

Peripheral Function	File Name	API Function Name
TIM Input Prescaler Mode	<Config_TIM*n_m*>.c	R_<Config_TIM*n_m*>_Create R_<Config_TIM*n_m*>_Start R_<Config_TIM*n_m*>_Stop R_<Config_TIM*n_m*>_SetDetectedEdge R_<Config_TIM*n_m*>_Software_Reset R_<Config_TIM*n_m*>_NEWVAL_TriggerOn R_<Config_TIM*n_m*>_ECNTOFL_TriggerOn R_<Config_TIM*n_m*>_CNTOFL_TriggerOn R_<Config_TIM*n_m*>_GPROFL_TriggerOn R_<Config_TIM*n_m*>_TODETOFL_TriggerOn R_<Config_TIM*n_m*>_GLITCHDETOFL_TriggerOn
	<Config_TIM*n_m*>_user.c	R_<Config_TIM*n_m*>_Create_UserInit r_<Config_TIM*n_m*>_Callback_GTM_Error r_<Config_TIM*n_m*>_share_interrupt
	<Config_TIM*n_m*>.h	—
TIM Input Event Mode	<Config_TIM*n_m*>.c	R_<Config_TIM*n_m*>_Create R_<Config_TIM*n_m*>_Start R_<Config_TIM*n_m*>_Stop R_<Config_TIM*n_m*>_Software_Reset R_<Config_TIM*n_m*>_NEWVAL_TriggerOn R_<Config_TIM*n_m*>_ECNTOFL_TriggerOn R_<Config_TIM*n_m*>_CNTOFL_TriggerOn R_<Config_TIM*n_m*>_GPROFL_TriggerOn R_<Config_TIM*n_m*>_TODETOFL_TriggerOn R_<Config_TIM*n_m*>_GLITCHDETOFL_TriggerOn
	<Config_TIM*n_m*>_user.c	R_<Config_TIM*n_m*>_Create_UserInit r_<Config_TIM*n_m*>_Callback_GTM_Error r_<Config_TIM*n_m*>_share_interrupt
	<Config_TIM*n_m*>.h	—

Table 2.19 Output File List (19/21)

Peripheral Function	File Name	API Function Name
TIM Pulse Integration Mode	<Config_TIM*n_m*>.c	R_<Config_TIM*n_m*>_Create R_<Config_TIM*n_m*>_Start R_<Config_TIM*n_m*>_Stop R_<Config_TIM*n_m*>_Software_Reset R_<Config_TIM*n_m*>_NEWVAL_TriggerOn R_<Config_TIM*n_m*>_ECNTOFL_TriggerOn R_<Config_TIM*n_m*>_CNTOFL_TriggerOn R_<Config_TIM*n_m*>_GPROFL_TriggerOn R_<Config_TIM*n_m*>_TODETOFL_TriggerOn R_<Config_TIM*n_m*>_GLITCHDETOFL_TriggerOn
	<Config_TIM*n_m*>_user.c	R_<Config_TIM*n_m*>_Create_UserInit r_<Config_TIM*n_m*>_Callback_GTM_Error r_<Config_TIM*n_m*>_share_interrupt
	<Config_TIM*n_m*>.h	—
TIM PWM Measurement Mode	<Config_TIM*n_m*>.c	R_<Config_TIM*n_m*>_Create R_<Config_TIM*n_m*>_Start R_<Config_TIM*n_m*>_Stop R_<Config_TIM*n_m*>_Software_Reset R_<Config_TIM*n_m*>_NEWVAL_TriggerOn R_<Config_TIM*n_m*>_ECNTOFL_TriggerOn R_<Config_TIM*n_m*>_CNTOFL_TriggerOn R_<Config_TIM*n_m*>_GPROFL_TriggerOn R_<Config_TIM*n_m*>_TODETOFL_TriggerOn R_<Config_TIM*n_m*>_GLITCHDETOFL_TriggerOn
	<Config_TIM*n_m*>_user.c	R_<Config_TIM*n_m*>_Create_UserInit r_<Config_TIM*n_m*>_Callback_GTM_Error r_<Config_TIM*n_m*>_share_interrupt
	<Config_TIM*n_m*>.h	—

Table 2.20 Output File List (20/21)

Peripheral Function	File Name	API Function Name
TIM Serial Shift Mode	<Config_TIM*n_m*>.c	R_<Config_TIM*n_m*>_Create R_<Config_TIM*n_m*>_Start R_<Config_TIM*n_m*>_Stop R_<Config_TIM*n_m*>_UpdateElapsedClock R_<Config_TIM*n_m*>_UpdateShiftClock R_<Config_TIM*n_m*>_UpdateCaptureSource R_<Config_TIM*n_m*>_UpdateTssmOut R_<Config_TIM*n_m*>_SetGPR0AsShadowRegister R_<Config_TIM*n_m*>_Software_Reset R_<Config_TIM*n_m*>_NEWVAL_TriggerOn R_<Config_TIM*n_m*>_ECNTOFL_TriggerOn R_<Config_TIM*n_m*>_CNTOFL_TriggerOn R_<Config_TIM*n_m*>_GPROFL_TriggerOn R_<Config_TIM*n_m*>_TODETOFL_TriggerOn R_<Config_TIM*n_m*>_GLITCHDETOFL_TriggerOn
	<Config_TIM*n_m*>_user.c	R_<Config_TIM*n_m*>_Create_UserInit r_<Config_TIM*n_m*>_Callback_GTM_Error r_<Config_TIM*n_m*>_share_interrupt
	<Config_TIM*n_m*>.h	—
ATOM Signal Output Mode Compare	<Config_ATOM*n_m*>.c	R_<Config_ATOMn_m>_Create R_<Config_ATOMn_m>_Start R_<Config_ATOMn_m>_Stop R_<Config_ATOMn_m>_Output_Start R_<Config_ATOMn_m>_Output_Stop R_<Config_ATOMn_m>_Channel_Reset R_<Config_ATOMn_m>_LateUpdate_Request
	<Config_ATOM*n_m*>_user.c	R_<Config_ATOM*n_m*>_Create_UserInit r_<Config_ATOM*n_m*>_Callback_Shared_IRQ*k*
	<Config_ATOM*n_m*>.h	—
ATOM Signal Output Mode Immediate	<Config_ATOM*n_m*>.c	R_<Config_ATOM*n_m*>_Create R_<Config_ATOM*n_m*>_Start R_<Config_ATOM*n_m*>_Stop R_<Config_ATOMn_m>_Output_Start R_<Config_ATOMn_m>_Output_Stop R_<Config_ATOMn_m>_Channel_Reset
	<Config_ATOM*n_m*>_user.c	R_<Config_ATOM*n_m*>_Create_UserInit
	<Config_ATOM*n_m*>.h	—

Table 2.21 Output File List (21/21)

Peripheral Function	File Name	API Function Name
ATOM Signal Output Mode PWM	<Config_ATOM*n_m*>.c	R_<Config_ATOMn_m>_Create R_<Config_ATOMn_m>_Start R_<Config_ATOMn_m>_Stop R_<Config_ATOMn_m>_Output_Start R_<Config_ATOMn_m>_Output_Stop R_<Config_ATOMn_m>_Channel_Reset
	<Config_ATOM*n_m*>_user.c	R_<Config_ATOM*n_m*>_Create_UserInit r_<Config_ATOMn_m>_Callback_Shared_IRQk
	<Config_ATOM*n_m*>.h	–
ATOM Signal Output Mode Serial	<Config_ATOM*n_m*>.c	R_<Config_ATOM*n_m*>_Create R_<Config_ATOM*n_m*>_Start R_<Config_ATOM*n_m*>_Stop R_<Config_ATOMn_m>_Output_Start R_<Config_ATOMn_m>_Output_Stop R_<Config_ATOMn_m>_Channel_Reset
	<Config_ATOM*n_m*>_user.c	R_<Config_ATOM*n_m*>_Create_UserInit r_<Config_ATOMn_m>_Callback_Shared_IRQk
	<Config_ATOM*n_m*>.h	–
Dead Time Module	<Config_CDTM*n_m*>.c	R_<Config_CDTM*n_m*>_Create
	<Config_CDTM*n_m*>_user.c	R_<Config_CDTM*n_m*>_Create_UserInit
	<Config_CDTM*n_m*>.h	–
DTS Controller	<Config_DTS*n*>.c	R_<Config_DTS*n*>_Create R_<Config_DTS*n*>_Start R_<Config_DTS*n*>_Stop R_<Config_DTS*n*>_Set_SoftwareTrigger
	<Config_DTS*n*>_user.c	R_<Config_DTS*n*>_Create_UserInit r_<Config_DTS*n*>_Callback_Dts*g*_Transfer_End r_<Config_DTS*n*>_Callback_Dts*g*_Transfer_Count_Match r_<Config_DTS*n*>_Callback_PE*k*_Transfer_Error r_<Config_DTSn>_Callback_Transfer_Error
	<Config_DTS*n*>.h	–
ADC Boundary Flag Generator	<Config_ABFG>.c	R_<Config_ABFG>_Create R_<Config_ABFG>_Start R_<Config_ABFG>_Stop
	<Config_ABFG>_user.c	R_<Config_ABFG>_Create_UserInit r_<Config_ABFG>_boundary_flag_pulse_*w*_interrupt
	<Config_ABFG>.h	–

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.

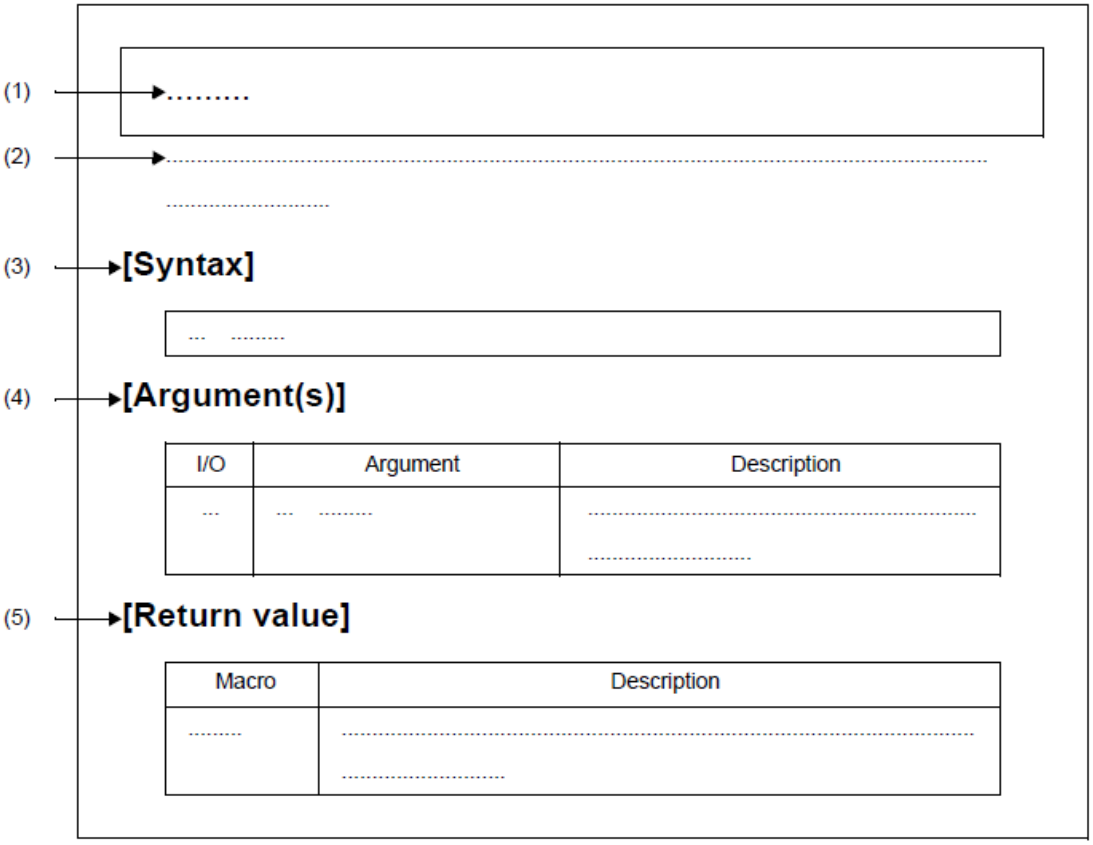
- Names of elements in enumeration specifiers “enum”
  These are in all-capital letters.

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.

I/O	Argument	Descripton
(a)	(b)	(c)

(a) I/O

Argument classification

I … Input argument

O … Output argument

(b) Argument

Argument data type

(c) Description

Description of argument

(5) [Return value]

API function return value is explained in the following format.

Macro	Description
(a)	(b)

(a) Macro

Macro of return value

(b) Description

Description of return value

3.2.1 General

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

Table 3.1.1 API Functions: [General] (1/2)

API Function Name	Description
main	This is a main function
R_MAIN_UserInit	Performs user-defined initialization
R_Systeminit	Performs initialization necessary to control the various peripheral functions
R_CGC_Create	Performs initialization required to control the clock generator (include reset function, on-chip debug function, etc.)
R_CGC_Create_UserInit	Performs user-define initialization relating to the clock generator (Include reset function, on-chip debug function, etc.)
R_Interrupt_Create	(Reserved function)
R_Pins_Create	This is a reference for setting the pin function
R_Interrupt_Initialize_ForPE	Initializes all interrupt of all used modules for PE*k*
R_ADC_SyncStart	Simultaneously starts A/D conversion for each scan group of all ADC units
R_ADC_TimerSyncStart	Simultaneously start A/D timer count operation of all ADC units
R_ATOM*n*_Start_Interrupt_IRQ*k*	Starts INTGTMA0ATOM*nk* interrupt
R_ATOM*n*_Stop_Interrupt_IRQ*k*	Stops INTGTMA0ATOM*nk* interrupt
r_atom*n*_irq*k*_interrupt	Performs processing in response to the INTGTMA0ATOM*nk* interrupt
R_SDMAC_Start_Error_Interrupt_PE*k*	Starts DMAC address error interrupt for PE*k*
R_SDMAC_Stop_Error_Interrupt_PE*k*	Stops DMAC address error interrupt for PE*k*
R_DMA_Suspend	Suspends DMA transfer for all channels
R_DMA_Resume	Resumes DMA transfer for all channels
R_PDMAn_Suspend	Suspends DMA transfer for all channels
R_PDMAn_Resume	Resumes DMA transfer for all channels
R_DMAC_Create	Initialize DMAC module
R_DMA_Start_Transfer_Error_Interrupt	Starts DMA transfer error interrupt in all DMAC and DTS channels
R_DMA_Stop_Transfer_Error_Interrupt	Stops DMA transfer error interrupt in all DMAC and DTS channels
r_sdmac_address_error_interrupt_pe*k*	Performs processing in response to the SDMAC address error interrupt for Pe*k*
r_dmac_error_interrupt_pe*k*	Performs processing in response to the DMA transfer error interrupt for PE1
R_DTS_Suspend	Suspends DTS transfer
R_DTS_Resume	Resumes DTS transfer
R_DTS*g*_Start_Transfer_End_Interrupt	Clears DTS channel *g* transfer end interrupt request and enable operation
R_DTS*g*_Stop_Transfer_End_Interrupt	Disables DTS channel *g* transfer end interrupt operation and clear request

Table 3.1.2 API Functions: [General] (2/2)

API Function Name	Description
R_DTS*g*_Start_Transfer_Count_Match_Interrupt	Clears DTS channel *g* transfer count match interrupt request and enable operation
R_DTS*g*_Stop_Transfer_Count_Match_Interrupt	Disables DTS channel *g* transfer count match interrupt operation and clear request
R_DTS_Start_PE*k*_Transfer_Error_Interrupt	Clears DTS transfer error interrupt request and enable operation for PE*k*
R_DTS_Stop_PE*k*_Transfer_Error_Interrupt	Disables DTC transfer error interrupt operation and clear request for PE*k*
r_dts*g*_transfer_end_interrupt	Performs processing in response to the DTS channel *g* transfer end interrupt
r_dts*g*_transfer_count_match_interrupt	Performs processing in response to the DTS channel *g* transfer count match interrupt
r_dts_transfer_error_interrupt_pe*k*	Performs processing in response to the DTS transfer error interrupt for PE*k*
R_GTM_Start_Interrupt_Error	Enables GTM error interrupt
R_GTM_Stop_Interrupt_Error	Disables GTM error interrupt
r_gtm_error_interrupt	Performs processing in response to the GTM error interrupt
R_MSPI_Master_Create	Performs initialization for all MSPI channels
R_MSPI_Start_Interrupt_MSPI*n*TX	Enables MSPI*n* transmit interrupt
R_MSPI_Stop_Interrupt_MSPI*n*TX	Disables MSPI*n* transmit interrupt
R_MSPI_Start_Interrupt_MSPI*n*RX	Enables MSPI*n* receive interrupt
R_MSPI_Stop_Interrupt_MSPI*n*RX	Disables MSPI*n* receive interrupt
R_MSPI_Start_Interrupt_MSPI*n*FE	Enables MSPI*n* frame end interrupt
R_MSPI_Stop_Interrupt_MSPI*n*FE	Disables MSPI*n* frame end interrupt
R_MSPI_Start_Interrupt_MSPI*n*ERR	Enables MSPI*n* error interrupt
R_MSPI_Stop_Interrupt_MSPI*n*ERR	Disables MSPI*n* error interrupt
r_mspi*n*_interrupt_send	Performs processing in response to the MSPI Master communication interrupt
r_mspi*n*_interrupt_receive	Performs processing in response to the MSPI*n* reception interrupt
r_mspi*n*_interrupt_frameend	Performs processing in response to the MSPI*n* frameend interrupt
r_mspi*n*_interrupt_error	Performs processing in response to the MSPI*n* error interrupt

main

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]

```
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]

```
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-define initialization relating to the clock generator (include reset function, on-chip debug function, etc.).

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

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_Interrupt_Initialize_ForPE

Performs initialization of all interrupts for each CPU core.

Remark This API function is called by R_Systeminit.

[Syntax]

```
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]

```
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]

```
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

Suspends DMA transfer for all channels.

Remark This API function is called by user.

[Syntax]

```
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]

```
void R_PDMAn_Suspend(void)
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_PDMA*n*_Resume

Resumes DMA transfer for all channels of unit *n*.

Remark This API function is called by user.

[Syntax]

```
void R_PDMAn_Resume(void)
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC_Start_Error_Interrupt_PE*k*

Starts SDMAC address error interrupt for PE*k*.

Remark This API function is called by user.

[Syntax]

```
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_PE*k*

Stops SDMAC address error interrupt for PE*k*.

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]

```
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_pe*k*

Performs processing in response to the SDMAC address error interrupt.

[Syntax]

```
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_pe*k*

Performs processing in response to the DMA transfer error interrupt for PE*k*

[Syntax]

```
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

Suspends DTS transfer.

Remark This API function is called by user.

[Syntax]

```
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]

```
void R_DTS_Resume(void)
```

[Argument(s)]

None.

[Return value]

None.

R_DTS*g*_Start_Transfer_End_Interrupt

Starts DTS channel *g* transfer end interrupt.

Remark This API function is called by user.

[Syntax]

```
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_DTS*g*_Stop_Transfer_End_Interrupt

Stops DTS channel *g* transfer end interrupt.

Remark This API function is called by user.

[Syntax]

```
void R_DTSg_Stop_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_DTS*g*_Start_Transfer_Count_Match_Interrupt

Starts DTS channel *g* transfer count match interrupt.

Remark This API function is called by user.

[Syntax]

```
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*g*_Stop_Transfer_Count_Match_Interrupt

Stops DTS channel *g* transfer count match interrupt.

Remark This API function is called by user.

[Syntax]

```
void R_DTSg_Stop_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_Start_PE*k*_Transfer_Error_Interrupt

Starts DTS transfer error interrupt for PE*k*.

Remark This API function is called by user.

[Syntax]

```
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_PE*k*_Transfer_Error_Interrupt

Stops DTS transfer error interrupt.

Remark This API function is called by user.

[Syntax]

```
void R_DTS_Stop_PEk_Transfer_Error_Interrupt(void)
```

Remark *k* is the CPU core number.

[Argument(s)]

None.

[Return value]

None.

r_dts*g*_transfer_end_interrupt

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

[Syntax]

void r_dts*g*_transfer_end_interrupt (void)

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

[Argument(s)]

None.

[Return value]

None.

r_dts*g*_transfer_count_match_interrupt

Performs processing in response to the DTS channel *g* transfer count match interrupt.

[Syntax]

```
void r_dtsg_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_pe*k*

Performs processing in response to the DTS transfer error interrupt for PE*k*.

[Syntax]

```
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_TIM*n_m*>_Start(TIM Gated Periodic Sampling Mode), R_<Config_TIM*n_m*>_Start(TIM Input Prescaler Mode), R_<Config_TIM*n_m*>_Start(TIM input event mode), R_<Config_TIM*n_m*>_Start(TIM pulse integration mode), R_<Config_TIM*n_m*>_Start(TIM pwm measurement mode), R_<Config_TIM*n_m*>_Start(TIM serial shift mode).

[Syntax]

```
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 functions is called indirectly by R_<Config_TIM*n*>_Stop(TIM Bit Compression Mode), R_<Config_TIM*n_m*>_Stop(TIM Gated Periodic Sampling Mode), R_<Config_TIM*n_m*>_Stop(TIM Input Prescaler Mode), R_<Config_TIM*n_m*>_Stop(TIM input event mode), R_<Config_TIM*n_m*>_Stop(TIM pulse integration mode), R_<Config_TIM*n_m*>_Stop(TIM pwm measurement mode), R_<Config_TIM*n_m*>_Stop(TIM serial shift mode)..

[Syntax]

```
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]

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

[Argument(s)]

None.

[Return value]

None.

R_ATOM*n*_Start_Interrupt_IRQ*k*

Starts INTGTMA0ATOM*nk* interrupt.

Remark This API function is called indirectly by R_<Config_ATOM*n*_*m*>_Start(ATOM Signal Output Mode Compare), R_<Config_ATOM*n*_*m*>_Start(ATOM Signal Output Mode PWM), R_<Config_ATOM*n*_*m*>_Start(ATOM Signal Output Mode Serial) function.

[Syntax]

```
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_ATOM*n*_Stop_Interrupt_IRQ*k*

Stops INTGTMA0ATOM*nk* interrupt.

Remark This API function is called indirectly by R_<Config_ATOM*n*_*m*>_Stop(ATOM Signal Output Mode Compare), R_<Config_ATOM*n*_*m*>_Stop(ATOM Signal Output Mode PWM), R_<Config_ATOM*n*_*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_atom*n*_irq*k*_interrupt

Performs processing in response to the INTGTMA0ATOM*nk* interrupt.

[Syntax]

```
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 functions is called by R_Systeminit function.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_MSPI_Start_Interrupt_MSPI*n*TX

Enables MSPI*n* transmit interrupt

Remark This API functions is called by user.

[Syntax]

```
void R_MSPI_Start_Interrupt_MSPInTX(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_MSPI_Stop_Interrupt_MSPI*n*TX

Disables MSPI*n* transmit interrupt

Remark This API functions is called by user.

[Syntax]

```
void R_MSPI_Stop_Interrupt_MSPInTX(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_MSPI_Start_Interrupt_MSPI*n*RX

Enables MSPI*n* receive interrupt

Remark This API functions is called by user.

[Syntax]

```
void R_MSPI_Start_Interrupt_MSPInRX(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_MSPI_Stop_Interrupt_MSPI*n*RX

Disables MSPI*n* receive interrupt

Remark This API functions is called by user.

[Syntax]

```
void R_MSPI_Stop_Interrupt_MSPInRX(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_MSPI_Start_Interrupt_MSPI*n*FE

Enables MSPI*n* frame end interrupt

Remark This API functions is called by user.

[Syntax]

```
void R_MSPI_Start_Interrupt_MSPInFE(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_MSPI_Stop_Interrupt_MSPI*n*FE

Disables MSPI*n* frame end interrupt

Remark This API functions is called by user.

[Syntax]

```
void R_MSPI_Stop_Interrupt_MSPInFE(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_MSPI_Start_Interrupt_MSPI*n*ERR

Enables MSPI*n* error interrupt

Remark This API functions is called by user.

[Syntax]

```
void R_MSPI_Start_Interrupt_MSPInERR(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_MSPI_Stop_Interrupt_MSPI*n*ERR

Disables MSPI*n* error interrupt

Remark This API functions is called by user.

[Syntax]

```
void R_MSPI_Stop_Interrupt_MSPInERR(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_mspi*n*_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*n*_interrupt_receive

Performs processing in response to the MSPI*n* reception interrupt.

[Syntax]

```
void    r_mspin_interrupt_receive(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_mspi*n*_interrupt_frameend

Performs processing in response to the MSPI*n* frameend interrupt.

[Syntax]

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

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_mspi*n*_interrupt_error

Performs processing in response to the MSPI*n* error interrupt.

[Syntax]

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

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_Interrupt_Create

Reserved function (Processing code is not output).

[Syntax]

```
void    R_Interrupt_Create(void);
```

[Argument(s)]

None.

[Return value]

None.

R_Pins_Create

This is a reference for setting the pin function (Other than I/O port).

For the pin function set on the [Pins] tab of the Smart Configurator, the setting code is output as a reference. Because it is a reference, it is not called by any function.

[Syntax]

```
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)

API Function Name	Description
R_<Config_ADC*Xn*>_Create	Performs initialization necessary to control the group scan mode A/D Converter functions
R_<Config_ADC*Xn*>_Halt	Halts A/D converter
R_<Config_ADC*Xn*>_SelfDiagnostic_VoltageCircuitOn	Turns on the self-diagnostic voltage circuit or updates the reference voltage.
R_<Config_ADC*Xn*>_SelfDiagnostic_VoltageCircuitOff	Turns off the self-diagnostic voltage circuit.
R_<Config_ADC*Xn*>_ScanGroup*x*_Start	Starts A/D converter of scan group
R_<Config_ADC*Xn*>_ScanGroup*x*_OperationOn	Starts scan group scan
R_<Config_ADC*Xn*>_ScanGroup*x*_OperationOff	Stops scan group scan
R_<Config_ADC*Xn*>_ScanGroup*x*_GetResult	Reads the A/D conversion results of scan group
R_<Config_ADC*Xn*>_ScanGroup*x*_GetPWMDiagResult	Gets A/D conversion results for ADC scan group (PWM-Diag).
R_<Config_ADC*Xn*>_ScanGroup*x*_GetFloatingPointResult	Gets A/D conversion result in floating point format for ADC scan group
R_<Config_ADC*Xn*>_SetMultiplexerCommand	Sets multiplexer command for ADC.
R_<Config_ADC*Xn*>_StrongPullDown_Start	Enables strong pull-down of the target physical channel.
R_<Config_ADC*Xn*>_StrongPullDown_Stop	Disables strong pull-down of the target physical channel.
R_<Config_ADC*Xn*>_TH_Groupk_Start	Starts T&H group hold
R_<Config_ADC*Xn*>_TH_Sampling_Start	Starts of sampling to all TH.
R_<Config_ADC*Xn*>_TH_Stop	Stops all T&H operation
R_<Config_ADC*Xn*>_SetMultiplexerCommand	Sets multiplexer command for ADC
R_<Config_ADC*Xn*>_ScanGroup*x*_TimerStart	Starts timer of ADC scan group
R_<Config_ADC*Xn*>_ScanGroup*x*_TimerStop	Stops timer of ADC scan group
R_<Config_ADC*Xn*>_VoltageDivider_Start	Enables voltage monitoring voltage divider
R_<Config_ADC*Xn*>_VoltageDivider_Stop	Disables voltage monitoring voltage divider
R_<Config_ADC*Xn*>_StrongPullDown_Start	Enables strong pulldown for physical channel which has strong pull-down function
R_<Config_ADC*Xn*>_StrongPullDown_Stop	Disables strong pulldown for physical channel which has strong pull-down function
R_<Config_ADC*Xn*>_SGDiag_Start	Enables A/D conversion of Diag scan groups
R_<Config_ADC*Xn*>_SGDiag_OperationOn	Starts A/D conversion of Diag scan groups
R_<Config_ADC*Xn*>_SGDiag_OperationOff	Stops A/D conversion of Diag scan groups
R_<Config_ADC*Xn*>_SGDiag_GetResult	Gets A/D conversion results for Diag scan groups
R_<Config_ADC*Xn*>_SGDiag_GetSubtractionResult	Gets all subtraction results for Diag scan groups

Table 3.3.2 API Functions: [A/D Converter] (2/2)

API Function Name	Description
R_<Config_ADC*Xn*>_ASF_Start	Starts of ASF accumulation counter
R_<Config_ADC*Xn*>_ASF_Stop	Stops of ASF accumulation counter
R_<Config_ADC*Xn*>_ASF_GetResult	Gets accumulation result by accumulation channel
R_<Config_ADC*Xn*>_ASF_GetRealTimeResult	Gets accumulation intermediate(real time) result by accumulation channel
R_<Config_ADC*Xn*>_Create_UserInit	Performs user-defined initialization relating to the group scan mode ADCA functions
r_<Config_ADC*Xn*>_error_interrupt	Performs processing in response to the A/D error interrupt
r_<Config_ADC*Xn*>_scan_group*x*_end_interrupt	Performs processing in response to the scan group end interrupt
r_<Config_ADC*Xn*>_monitor_error_interrupt	Performs processing in response to the monitor error interrupt
r_<Config_ADC*Xn*>_sg_diag_end_interrupt	Performs processing in response to SG-Diag end interrupt
r_<Config_ADC*Xn*>_mpx_request_interrupt	Performs processing in response to MPX DMA trigger request interrupt
r_<Config_ADC*Xn*>_asf_channelk_end_interrupt	Performs processing in response to accumulation end interrupt

R_<Config_ADC*Xn*>_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 ADC*X* is ADC name (ADCA, ADCC, ADCJ, ADCK), *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_ADC*Xn*>_Halt

Halts A/D converter.

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ADC*Xn*>_SelfDiagnostic_VoltageCircuitOn

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

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ADC*Xn*>_SelfDiagnostic_VoltageCircuitOff

Turns off the self-diagnostic voltage circuit.

[Syntax]

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

Remark ADC*X* 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]

```
void    R_<Config_ADCXn>_ScanGroupx_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_ADC*Xn*>_ScanGroup*x*_OperationOn

Starts scan group scan by using a software trigger.

[Syntax]

```
void    R_<Config_ADCXn>_ScanGroupx_OperationOn(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_ADC*Xn*>_ScanGroup*x*_OperationOff

Stops scan group scan by using a software trigger.

[Syntax]

```
void    R_<Config_ADCXn>_ScanGroupx_OperationOff(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_ADC*Xn*>_ScanGroup*x*_GetResult

Reads the A/D conversion results of scan group.

[Syntax]

MD_STATUS R_<Config_ADC*Xn*>_ScanGroup*x*_GetResult(uint16_t * const *buffer*, uint8_t *buffer_size*);

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

[Argument(s)]

I/O	Argument(s)	Description
O	uint16_t * const *buffer*	Pointer to area in which to store the results of A/D conversion
I	uint8_t *buffer_size*	The size of buffer

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	The actual converted data number is beyond the *buffer_size*

R_<Config_ADC*Xn*>_ScanGroup*x*_GetPWMDiagResult

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

[Syntax]

MD_STATUS R_<Config_ADC*Xn*>_ScanGroup*x*_GetPWMDiagResult(uint16_t * const *buffer*);

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

[Argument(s)]

I/O	Argument(s)	Description
O	uint16_t * const *buffer*	Pointer to area in which to store the results of A/D conversion

[Return value]

Macro	Description
MD_OK	Normal completion

R_<Config_ADC*Xn*>_ScanGroup*x*_GetFloatingPointResult

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

[Syntax]

MD_STATUS R_<Config_ADC*Xn*>_ScanGroup*x*_GetFloatingPointResult(uint32_t * const buffer, uint8_t *buffer_size*);

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

[Argument(s)]

I/O	Argument(s)	Description
O	uint32_t * const buffer	Pointer to area in which to store the results of A/D conversion
I	uint8_t *buffer_size*	The size of buffer

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	The actual converted data number is beyond the *buffer_size*

R_<Config_ADC*Xn*>_SetMultiplexerCommand

Sets multiplexer command for ADC.

[Syntax]

```
void    R_<Config_ADCXn>_SetMultiplexerCommand;
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ADC*Xn*>_TH_Group*k*_Start

Starts T&H group hold.

[Syntax]

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

Remark ADC*X* 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_ADC*Xn*>_TH_Sampling_Start

Starts T&H sampling.

[Syntax]

```
void    R_<Config_ADCXn>_TH_Sampling_Start(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ADC*Xn*>_TH_Stop

Stops Stops all T&H operation.

[Syntax]

```
void    R_<Config_ADCXn>_TH_Stop(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ADC*Xn*>_SetMultiplexerCommand

Sets multiplexer command for ADC.

[Syntax]

```
void    R_<Config_ADCXn>_ SetMultiplexerCommand(uint8_t cmdData);
```

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

[Argument(s)]

I/O	Argument(s)	Description
I	uint8_t cmdData	Multiplexer command

[Return value]

None.

R_<Config_ADC*Xn*>_ScanGroup*x*_TimerStart

Starts timer of ADC scan group.

[Syntax]

```
void    R_<Config_ADCXn>_ScanGroupx_TimerStart(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_ADC*Xn*>_ScanGroup*x*_TimerStop

Stops timer of ADC scan group.

[Syntax]

```
void    R_<Config_ADCXn>_ScanGroupx_TimerStop(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_ADC*Xn*>_VoltageDivider_Start

Enables voltage monitoring voltage divider.

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ADC*Xn*>_VoltageDivider_Stop

Disables voltage monitoring voltage divider.

[Syntax]

```
void    R_<Config_ADCXn>_VoltageDivider_Stop(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ADC*Xn*>_StrongPullDown_Start

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

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ADC*Xn*>_StrongPullDown_Stop

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

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ADC*Xn*>_SGDiag_Start

Enables A/D conversion of Diag scan groups

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ADC*Xn*>_SGDiag_OperationOn

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

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ADC*Xn*>_SGDiag_OperationOff

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

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ADC*Xn*>_SGDiag_GetResult

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

[Syntax]

MD_STATUS R_<Config_ADC*Xn*>_SGDiag_GetResult(uint16_t * const *buffer*, uint8_t *buffer_size*);

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

[Argument(s)]

I/O	Argument(s)	Description
O	uint16_t * const *buffer*	Pointer to area in which to store the results of A/D conversion
I	uint8_t *buffer_size*	The size of buffer

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	The actual converted data number is beyond the *buffer_size*

R_<Config_ADC*Xn*>_SGDiag_GetSubtractionResult

Gets all subtraction results for Diag scan groups.

[Syntax]

MD_STATUS R_<Config_ADC*Xn*>_SGDiag_GetSubtractionResult(uint32_t * const *buffer*, uint8_t *buffer_size*);

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

[Argument(s)]

I/O	Argument(s)	Description
O	uint32_t * const buffer	Pointer to area in which to store the results of A/D conversion
I	uint8_t *buffer_size*	The size of buffer

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	The actual converted data number is beyond the *buffer_size*

R_<Config_ADC*Xn*>_ASF_Start

Starts ASF accumulation counter.

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ADC*Xn*>_ASF_Stop

Stops ASF accumulation counter.

[Syntax]

```
void    R_<Config_ADCXn>_ASF_Stop(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ADC*Xn*>_ASF_GetResult

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

[Syntax]

MD_STATUS R_<Config_ADC*Xn*>_ASF_GetResult(uint8_t *channel*, uint32_t* const *data*);

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

[Argument(s)]

I/O	Argument(s)	Description
I	uint8_t *channel*	Accumulation channel i (i=0 to 15)
O	uint32_t * const *data*	Data pointer to area in which to store an accumulation data

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	The *channel* is out of the accumulation channel range (0 to 15)

R_<Config_ADC*Xn*>_ASF_GetRealTimeResult

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

[Syntax]

MD_STATUS R_<Config_ADC*Xn*>_ASF_GetRealTimeResult(uint8_t *channel*, uint32_t* const *data*, uint16_t* const *counter*);

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

[Argument(s)]

I/O	Argument(s)	Description
I	uint8_t *channel*	Accumulation channel *i* (*i*=0 to 15)
O	uint32_t * const *data*	Data pointer to area in which to store an accumulation data
O	uint16_t* const *counter*	Counter pointer to area in which to store accumulation counter value

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	The *channel* is out of the accumulation channel range (0 to 15)

R_<Config_ADC*Xn*>_Create_UserInit

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

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

[Syntax]

```
void R_<Config_ADCXn>_Create_UserInit(void);
```

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_ADC*Xn*>_error_interrupt

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

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_ADC*Xn*>_scan_group*x*_end_interrupt

Performs processing in response to the scan group end interrupt.

[Syntax]

```
void    r_<Config_ADCXn>_scan_groupx_end_interrupt(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_ADC*Xn*>_monitor_error_interrupt

Performs processing in response to the monitor error interrupt.

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_ADC*Xn*>_sg_diag_end_interrupt

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

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_ADC*Xn*>_mpx_request_interrupt

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

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_ADC*Xn*>_asf_channel*k*_end_interrupt

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

[Syntax]

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

Remark ADC*X* 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]

API Function Name	Description
R_<Config_CSI*Xn*>_Create	Performs initialization necessary to control the CSI slave functions
R_<Config_CSI*Xn*>_Start	Enables CSI
R_<Config_CSI*Xn*>_Stop	Disables CSI
R_<Config_CSI*Xn*>_Send	Starts data transmission
R_<Config_CSI*Xn*>_Receive	Starts data reception
R_<Config_CSI*Xn*>_Send_Receive	Starts data transmission and reception
R_<Config_CSI*Xn*>_Create_UserInit	Performs user-defined initialization relating to the CSI slave functions
r_<Config_CSI*Xn*>_interrupt_send	Processing is performed according to the generation of the send status interrupt
r_<Config_CSI*Xn*>_interrupt_receive	Processing is performed according to the generation of the receive status interrupt
r_<Config_CSI*Xn*>_interrupt_error	Processing is performed according to the occurrence of a communication error interrupt
r_<Config_CSI*Xn*>_callback_sendend	Processing is performed according to the generation of the send status interrupt
r_<Config_CSI*Xn*>_callback_receiveend	Processing is performed according to the generation of the receive status interrupt
r_<Config_CSI*Xn*>_callback_error	Processing is performed according to the occurrence of a communication error interrupt

R_<Config_CSI*Xn*>_Create

Performs initialization necessary to control the CSI slave 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*Xn*>_Start

Enables CSI.

[Syntax]

```
void    R_<Config_CSIXn>_Start(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_CSI*Xn*>_Stop

Disables CSI.

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_CSI*Xn*>_Send

Starts data transmission.

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

Remark2. R_<Config_CSI*Xn*>_Start must be called before this API function is called.

[Syntax]

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

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

[Argument(s)]

I/O	Argument(s)	Description
I	uint16_t* const *tx_buf*	Pointer to a buffer storing the transmission data
I	uint16_t *tx_num*	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_<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 *rx_num*, and stores the data in the buffer specified by the argument *rx_buf*.

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

[Syntax]

MD_STATUS R_<Config_CSI*Xn*>_Receive(uint16_t* const *rx_buf*, uint16_t *rx_num*);

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

[Argument(s)]

I/O	Argument(s)	Description
O	uint16_t* const *rx_buf*	Pointer to a buffer to store the received data
I	uint16_t *rx_num*	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion.
MD_ARGERROR	Invalid argument specification

R_<Config_CSI*Xn*>_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_CSI*Xn*>_Start must be called before this API function is called.

[Syntax]

MD_STATUS R_<Config_CSI*Xn*>_Send_Receive(uint16_t* const *tx_buf*, uint16_t *rx_num*, uint16_t* const *rx_buf*);

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

[Argument(s)]

I/O	Argument(s)	Description
I	uint16_t* const *tx_buf*	Pointer to a buffer storing the transmission data which count is same as received data
I	uint16_t *rx_num*	Total amount of data to receive
O	uint16_t* const *rx_buf*	Pointer to a buffer to store the received data

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid specification of *rx_num*

R_<Config_CSI*Xn*>_Create_UserInit

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

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

[Syntax]

```
void R_<Config_CSIXn>_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*Xn*>_interrupt_send

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

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_CSI*Xn*>_interrupt_receive

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

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_CSI*Xn*>_interrupt_error

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

[Syntax]

```
void    r_<Config_CSIXn>_interrupt_error(void);
```

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_CSI*Xn*>_callback_sendend

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

Remark This API function is called as the callback routine of interrupt process r_<Config_CSI*Xn*>_interrupt_send corresponding to the CSIG communication interrupt.

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_CSI*Xn*>_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_CSI*Xn*>_interrupt_receive corresponding to the CSIG reception interrupt.

[Syntax]

```
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_CSI*Xn*>_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_CSI*Xn*>_interrupt_error the CSIG error interrupt.

[Syntax]

```
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)]

I/O	Argument(s)	Description
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

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]

API Function Name	Description
R_<Config_CSI*Xn*>_Create	Performs initialization necessary to control the CSI master functions
R_<Config_CSI*Xn*>_Start	Enables CSI
R_<Config_CSI*Xn*>_Stop	Prohibit CSI
R_<Config_CSI*Xn*>_Send	Starts data transmission
R_<Config_CSI*Xn*>_Receive	Starts data reception
R_<Config_CSI*Xn*>_Send_Receive	Starts data transmission and reception
R_<Config_CSI*Xn*>_Create_UserInit	Performs user-defined initialization relating to the CSI master functions
r_<Config_CSI*Xn*>_interrupt_send	Processing is performed according to the generation of the send status interrupt
r_<Config_CSI*Xn*>_interrupt_receive	Processing is performed according to the generation of the receive status interrupt
r_<Config_CSI*Xn*>_interrupt_error	Processing is performed according to the occurrence of a communication error interrupt
r_<Config_CSI*Xn*>_callback_sendend	Processing is performed according to the generation of the send status interrupt
r_<Config_CSI*Xn*>_callback_receiveend	Processing is performed according to the generation of the receive status interrupt
r_<Config_CSI*Xn*>_callback_error	Processing is performed according to the occurrence of a communication error interrupt

R_<Config_CSI*Xn*>_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*Xn*>_Start

Enables CSI.

[Syntax]

```
void    R_<Config_CSIXn>_Start(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_CSI*Xn*>_Stop

Disables CSI.

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_CSI*Xn*>_Send

Starts data transmission.

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

Remark2 R_<Config_CSI*Xn*>_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)]

I/O	Argument(s)	Description
I	uint16_t* const *tx_buf*	Pointer to a buffer storing the transmission data
I	uint16_t *tx_num*	Total amount of data to send
I	uint32_t *chip_id*	Slave channel ID (Multiple choices are possible by using “|” operator) _CSIH_SELECT_CHIP_*n*: Channel n

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid specification of *tx_num*

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 *rx_num*, and stores the data in the buffer specified by the argument *rx_buf*.

Remark2 Starts after this API function is called, and R_<Config_CSI*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 *n* is the channel number.

CSIH case

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

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	uint16_t* const *rx_buf*	Pointer to a buffer to store the received data
I	uint16_t *rx_num*	Total amount of data to receive
I	uint32_t *chip_id*	Slave channel ID (Multiple choices are possible by using “|” operator) _CSIH_SELECT_CHIP_n: Channel n

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid specification of *rx_num*

R_<Config_CSI*Xn*>_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 t*x_num*, and stores the data in the buffer specified by the argument *rx_buf*.

Remark3 R_<Config_CSI*Xn*>_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)]

I/O	Argument(s)	Description
I	uint16_t* const *tx_buf*	Pointer to a buffer storing the transmission data
I	uint16_t *tx_num*	Total amount of data to send
O	uint16_t* const *rx_buf*	Pointer to a buffer to store the received data which count is same as sent data.
I	uint32_t *chip_id*	Slave channel: _CSIH_SELECT_CHIP_*n*: Channel n

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid specification of *tx_num*

R_<Config_CSI*Xn*>_Create_UserInit

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

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

[Syntax]

```
void R_<Config_CSIXn>_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*Xn*>_interrupt_send

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

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_CSI*Xn*>_interrupt_receive

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

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_CSI*Xn*>_interrupt_error

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

[Syntax]

```
void    r_<Config_CSIXn>_interrupt_error(void);
```

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_CSI*Xn*>_callback_sendend

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

Remark This API function is called as the callback routine of interrupt process r_<Config_CSI*Xn*>_interrupt_send corresponding to the CSIG communication interrupt.

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_CSI*Xn*>_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_CSI*Xn*>_interrupt_receive corresponding to the CSIG reception interrupt.

[Syntax]

```
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_CSI*Xn*>_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_CSI*Xn*>_interrupt_error corresponding to the CSIG error interrupt.

[Syntax]

```
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)]

I/O	Argument(s)	Description
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.)

r_cg_main.c

```
/* 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);
    while(transmitend_flag0 != 1);
    while(receiveend_flag0 != 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);
    while(transmitend_flag0 != 1);
    while(receiveend_flag0 != 1);
    transmitend_flag1 = 0;
    receiveend_flag1 = 0;
    transmitend_flag0 = 0;
    receiveend_flag0 = 0;
    while(1);
    /* End user code. Do not edit comment generated here */
}
```

Config_CSIH1_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_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 3.6 API Functions: [Interrupt]

API Function Name	Description
R_<Config_INTC>_Create	Performs initialization necessary to control the interrupt functions
R_<Config_INTC>_INTP*n*_Start	Enables the INTP*n* interrupts
R_<Config_INTC>_INTP*n*_Stop	Disables the INTP*n* interrupts
R_<Config_INTC>_IRQ*n*_Start	Enables the IRQ*n* interrupts
R_<Config_INTC>_IRQ*n*_Stop	Disables the IRQ*n* interrupts
R_<Config_INTC>_Create_UserInit	Performs user-defined initialization relating to the interrupt functions
r_<Config_INTC>_intp*n*_interrupt	Performs processing in response to the INTP*n* interrupt
r_<Config_INTC>_intpn_interrupt_pek	Performs processing in response to the INTP*n* interrupt for PE *k*
r_<Config_INTC>_irq*n*_interrupt	Performs processing in response to the IRQ*n* interrupt
r_<Config_INTC>_nmi_interrupt	Performs processing in response to the NMI interrupt
R_<Config_INTC>_NMI_Start	Enables NMI interrupt
R_<Config_INTC>_NMI_Stop	Disables NMI interrupt
R_<Config_INTC>_SINT*n*_Start	Enables the SINT*n* interrupt
R_<Config_INTC>_SINT*n*_Stop	Disables the SINT*n* interrupt
R_<Config_INTC>_SINT*n*_TriggerOn	Triggers SINT*n* interrupt
r_<Config_INTC>_sint*n*_interrupt_pe*k*	Performs processing in response to the SINT*n* interrupt for PE*k*

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>_INTP*n*_Start

Enables the INTP*n* interrupts.

[Syntax]

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

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_INTC>_INTP*n*_Stop

Disables the INTP*n* 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>_IRQ*n*_Start

Enables the IRQ*n* interrupts.

[Syntax]

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

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_INTC>_IRQ*n*_Stop

Disables the IRQ*n* interrupts.

[Syntax]

```
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 functions is called as the R_<Config_INTC>_Create callback routine.

[Syntax]

```
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*n* interrupt.

[Syntax]

```
void    r_<Config_INTC>_intpn_interrupt(void);
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_INTC>_intp*n*_interrupt_pe*k*

Performs processing in response to the INTP*n* interrupt for PE *k*.

[Syntax]

```
void    r_<Config_INTC>_intpn_interrupt_pek(void);
```

Remark *n* is the interrupt factor number, *k* is the CPU core number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_INTC>_irq*n*_interrupt

Performs processing in response to the Irq*n* interrupt.

[Syntax]

```
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]

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_INTC>_NMI_Start

Enables the NMI interrupts.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_INTC>_NMI_Stop

Disables the NMI interrupts.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_INTC>_SINT*n*_Start

Enables the SINT*n* interrupts.

[Syntax]

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

Remark *n* is the software interrupt number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_INTC>_SINT*n*_Stop

Disables the SINT*n* interrupts.

[Syntax]

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

Remark *n* is the software interrupt number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_INTC>_SINT*n*_TriggerOn

Triggers the SINT*n* 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>_sint*n*_interrupt_pe*k*

Performs processing in response to the SINT*n* 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]

API Function Name	Description
R_<Config_TAU*Xn_m*>_Create	Performs initialization necessary to control the input interval timer functions
R_<Config_TAU*Xn_m*>_Start	Starts the count for channel *m*
R_<Config_TAU*Xn_m*>_Stop	Ends the count for channel *m*
R_<Config_TAU*Xn_m*>_Create_UserInit	Performs user-defined initialization relating to the input interval timer functions
r_<Config_TAU*Xn_m*>_interrupt	Performs processing in response to the timer interrupt
r_<Config_TAU*Xn_m*>_interrupt_pe*k*	Performs processing in response to the timer interrupt for PE*k*

R_<Config_TAU*Xn_m*>_Create

Performs initialization necessary to control the input interval timer functions.

Remark This API function is called by R_Systeminit function.

[Syntax]

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

Remark TAU*X* 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*>_Start

Starts the count for channel *m*.

[Syntax]

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

Remark TAU*X* 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]

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

Remark TAU*X* 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*>_Create_UserInit

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

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

[Syntax]

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

Remark TAU*X* 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

Performs processing in response to the timer interrupt.

[Syntax]

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

Remark TAU*X* 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_pe*k*

Performs processing in response to the timer interrupt for PE*k*.

[Syntax]

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

Remark TAU*X* 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]

API Function Name	Description
R_<Config_TAU*Xn_m*>_Create	Performs initialization necessary to control the input pulse interval measurement functions
R_<Config_TAU*Xn_m*>_Start	Starts the count for channel *m*
R_<Config_TAU*Xn_m*>_Stop	Ends the count for channel *m*
R_<Config_TAU*Xn_m*>_Get_PulseWidth	Reads the input pulse width of the timer
R_<Config_TAU*Xn_m*>_Create_UserInit	Performs user-defined initialization relating to the input pulse interval measurement functions
r_<Config_TAU*Xn_m*>_interrupt	Performs processing in response to the timer interrupt
r_<Config_TAU*Xn_m*>_interrupt_pe*k*	Performs processing in response to the timer interrupt for PE*k*

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]

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

Remark TAU*X* 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*>_Start

Starts the count for channel *m*.

[Syntax]

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

Remark TAU*X* 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]

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

Remark TAU*X* 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]

void R_<Config_TAU*Xn_m*>_*Get*_PulseWidth(uint32_t * const *width*);

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

[Argument(s)]

I/O	Argument(s)	Description
O	uint32_t * const *width*	Pointer to area in which to store the results of input pulse width

[Return value]

None.

R_<Config_TAU*Xn_m*>_Create_UserInit

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

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

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_TAU*Xn_m*>_interrupt

Performs processing in response to the timer interrupt.

[Syntax]

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

Remark TAU*X* 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_pe*k*

Performs processing in response to the timer interrupt for PE*k*.

[Syntax]

```
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]

API Function Name	Description
R_<Config_TAU*Xn_m*>_Create	Performs initialization necessary to control the interval timer functions
R_<Config_TAU*Xn_m*>_Start	Starts the count for channel *m*
R_<Config_TAU*Xn_m*>_Stop	Ends the count for channel *m*
R_<Config_TAU*Xn_m*>_Create_UserInit	Performs user-defined initialization relating to the interval timer functions
r_<Config_TAU*Xn_m*>_interrupt	Performs processing in response to the timer interrupt
r_<Config_TAU*Xn_m*>_interrupt_pe*k*	Performs processing in response to the timer interrupt for PE*k*

R_<Config_TAU*Xn_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*X* 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*>_Start

Starts the count for channel *m*.

[Syntax]

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

Remark TAU*X* 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]

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

Remark TAU*X* 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*>_Create_UserInit

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

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

[Syntax]

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

Remark TAU*X* 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

Performs processing in response to the timer interrupt.

[Syntax]

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

Remark TAU*X* 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_pe*k*

Performs processing in response to the timer interrupt for PE*k*.

[Syntax]

```
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]

API Function Name	Description
R_<Config_TAU*Xn*>_Create	Performs initialization necessary to control the triangle PWM output functions
R_<Config_TAU*Xn*>_Start	Starts timer counting
R_<Config_TAU*Xn*>_Stop	Stops timer counting
R_<Config_TAU*Xn*>_Create_UserInit	Performs user-defined initialization relating to the triangle PWM output functions
r_<Config_TAU*Xn*>_channelm_interrupt	Performs processing in response to the timer interrupt
r_<Config_TAU*Xn*>_channelm_interrupt_pek	Performs processing in response to the timer interrupt for PE*k*

R_<Config_TAU*Xn*>_Create

Performs initialization necessary to control the triangle PWM output functions.

Remark This API function is called by R_Systeminit function.

[Syntax]

```
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_TAU*Xn*>_Start

Starts timer counting.

[Syntax]

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

Remark TAU*X* 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]

```
void    R_<Config_TAUXn>_Stop(void);
```

Remark TAU*X* 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 triangle PWM output functions.

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

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_TAU*Xn*>_channel*m*_interrupt

Performs processing in response to the timer interrupt.

[Syntax]

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

Remark TAU*X* 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*>_channel*m*_interrupt_pe*k*

Performs processing in response to the timer interrupt for PE*k*.

[Syntax]

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

Remark TAU*X* 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.

3.2.10 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]

API Function Name	Description
R_<Config_TAU*Xn*>_Create	Performs initialization necessary to control the triangle PWM output with dead time functions
R_<Config_TAU*Xn*>_Start	Starts the count for channel *m*
R_<Config_TAU*Xn*>_Stop	Ends the count for channel *m*
R_<Config_TAU*Xn*>_Create_UserInit	Performs user-defined initialization relating to the triangle PWM output with dead time functions
r_<Config_TAU*Xn*>_channel*m*_interrupt	Performs processing in response to the timer interrupt
r_<Config_TAU*Xn*>_channel*m*_interrupt_pe*k*	Performs processing in response to the timer interrupt for PE*k*

R_<Config_TAU*Xn*>_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]

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

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_TAU*Xn*>_Start

Starts the counting.

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_TAU*Xn*>_Stop

Ends the counting.

[Syntax]

```
void    R_<Config_TAUXn>_Stop(void);
```

Remark TAU*X* 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 triangle PWM output with dead time functions.

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

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_TAU*Xn*>_channel*m*_interrupt

Performs processing in response to the timer interrupt.

[Syntax]

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

Remark TAU*X* 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*>_channel*m*_interrupt_pe*k*

Performs processing in response to the timer interrupt for PE*k*

[Syntax]

```
void    r_<Config_TAUXn>_channelm _interrupt_pek(void);
```

Remark TAU*X* 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]

API Function Name	Description
R_<Config_OSTM*n*>_Create	Performs initialization necessary to control the OS timer functions
R_<Config_OSTM*n*>_Start	Starts OS timer count
R_<Config_OSTM*n*>_Stop	Stop OS timer count
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
R_<Config_OSTM*n*>_Create_UserInit	Performs user-defined initialization relating to the OS timer functions
r_<Config_OSTM*n*>_interrupt	Performs processing in response to the OS timer interrupt

R_<Config_OSTM*n*>_Create

Performs initialization necessary to control the OS timer functions.

Remark This API function is called by R_Systeminit function.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_OSTM*n*>_Start

Starts OS timer count.

[Syntax]

```
void    R_<Config_OSTMn>_Start(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_OSTM*n*>_Stop

Stop OS timer count.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_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]

```
void    R_<Config_OSTMn_Set_CompareValue(uint32_t value);
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
I	uint32_t *value*	Start value of the down-counter or comparison value

[Return value]

None.

R_<Config_OSTM*n*>_Create_UserInit

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

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

[Syntax]

```
void R_<Config_OSTMn>_Create_UserInit(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_OSTM*n*>_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]

API Function Name	Description
R_<Config_PORT>_Create	Performs initialization necessary to control the I/O port functions
R_<Config_PORT>_Create_UserInit	Performs user-defined initialization relating to I/O port functions

R_<Config_PORT>_Create

Performs initialization necessary to control the I/O port.

Remark This API function is called by R_Systeminit function.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_PORT>_Create_UserInit

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

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

[Syntax]

```
void R_<Config_PORT>_Create_UserInit(void);
```

[Argument(s)]

None.

[Return value]

None.

3.2.13 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]

API Function Name	Description
R_<Config_TAU*Xn*>_Create	Performs initialization necessary to control the PWM output functions
R_<Config_TAU*Xn*>_Start	Starts timer counting
R_<Config_TAU*Xn*>_Stop	Stops timer counting
R_<Config_TAU*Xn*>_Create_UserInit	Performs user-defined initialization relating to the PWM output functions
r_<Config_TAU*Xn*>_channel*m*_interrupt	Performs processing in response to the timer interrupt
r_<Config_TAU*Xn*>_channel*m*_interrupt_pe*k*	Performs processing in response to the timer interrupt for PE*k*

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]

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

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_TAU*Xn*>_Start

Starts timer counting.

[Syntax]

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

Remark TAU*X* 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]

```
void    R_<Config_TAUXn>_Stop(void);
```

Remark TAU*X* 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 PWM output functions.

Remark This API functions is called as the R_<Config_TAU*Xn*>_*Create* callback routine.

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_TAU*Xn*>_channel*m*_interrupt

Performs processing in response to the timer interrupt.

[Syntax]

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

Remark TAU*X* 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*>_channel*m*_interrupt_pe*k*

Performs processing in response to the timer interrupt for PE*k*

[Syntax]

```
void    r_<Config_TAUXn>_channelm _interrupt_pek(void);
```

Remark TAU*X* 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]

API Function Name	Description
R_<Config_STBC>_Prepare_Stop_Mode	Performs user-defined processing relating to the preparation to start stand-by (STOP mode)
R_<Config_STBC>_Start_Stop_Mode	Starts stand-by (STOP mode)
R_<Config_STBC>_Prepare_Deep_Stop_Mode	Performs user-defined processing relating to the preparation to start stand-by (DeepSTOP mode)
R_<Config_STBC>_Start_Deep_Stop_Mode	Starts stand-by (DeepSTOP mode)
R_<Config_STBC>_Deep_Stop_Loop	Performs wait processing of stand-by (DeepSTOP mode)
R_<Config_STBC>_Prepare_Stop_Mode_Set_Peripheral	Performs user-defined processing relating to the preparation (stop peripheral) to start stand-by (STOP mode)
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)
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)
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)
R_<Config_STBC>_Prepare_Stop_Mode_Set_CPUCLK	Performs user-defined processing relating to the preparation (CPU clock setting) to start stand-by (STOP mode)
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)
R_<Config_STBC>_Prepare_Deep_Stop_Mode_Set_Interrupt	Performs user-defined processing relating to the preparation (interrupt control register) to start stand-by (DeepSTOP mode)
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)
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)
R_<Config_STBC>_Set_Module_*XXXX*_Standby_Mode	Set module standby mode by stopping clock connected to XXXX channel
R_<Config_STBC>_Cancel_Module_*XXXX*_Standby_Mode	Cancel module standby mode by supplying clock connected to XXXX channel
R_<Config_STBC>_Check_Module_*XXXX*_Idle_State	Check if module XXXX is in idle status before setting module standby mode

R_<Config_STBC>_Prepare_Stop_Mode

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

[Syntax]

```
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]

```
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]

```
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]

```
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]

```
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]

```
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]

```
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]

```
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]

```
void    R_<Config_STBC>_Prepare_Deep_Stop_Mode_Set_Peripheral(void);
```

[Argument(s)]

None.

[Return value]

None.

R_<Config_STBC>_Prepare_Deep_Stop_Mode_Set_Interrupt

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

[Syntax]

```
void    R_<Config_STBC>_Prepare_Deep_Stop_Mode_Set_Interrupt(void);
```

[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]

```
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]

```
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 3.14.1 XXXX module/function name

	XXXX name		XXXX name		XXXX name
1	RSCFD	10	MMCA	19	TAUD
2	FLXA	11	ENCA	20	TAUJ_ISO
3	GTM	12	PSI5	21	TPBA
4	ETNB	13	PSI5S	22	TSG3
5	RSENT	14	PWMD	23	OSTM
6	MSPI	15	RHSIF	24	ADCJ_AWO
7	RLIN3	16	RIIC	25	RTCA
8	ADCJ_ISO	17	SCI3	26	TAUJ_AWO
9	CXPI	18	TAPA		

[Argument(s)]

I/O	Argument(s)	Description
I	Uint8_t channel	XXXX channel

[Return value]

Macro	Description
MD_OK	Set module standby mode successfully
MD_BUSY1	Reset execution of module XXXX is being processed

R_<Config_STBC>_Cancel_Module_ XXXX_Standby_Mode

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

[Syntax]

```
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)]

I/O	Argument(s)	Description
I	Uint8_t channel	XXXX channel

[Return value]

Macro	Description
MD_OK	Set module standby mode successfully
MD_BUSY1	Reset execution of module XXXX is being processed

R_<Config_STBC>_Check_Module_XXXX_Idle_State

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

Remark This API is called by R_<Config_STBC>_Set_Module_XXXX_Standby_Mode.

[Syntax]

```
void R_<Config_STBC_Check_Module_XXXX_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.)

r_cg_main.c

```
void main(void)
{
    r_main_userinit();
    /* Start user code for main. Do not edit comment generated here */
    R_Config_INTC_INTP12_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]

API Function Name	Description
R_<Config_UART*n*>_Create	Performs initialization necessary to control the UART interface functions
R_<Config_UART*n*>_Start	Sets UART communication to standby mode
R_<Config_UART*n*>_Stop	Ends UART communication
R_<Config_UART*n*>_Send	Starts UART data transmission
R_<Config_UART*n*>_Receive	Starts UART data reception
R_<Config_UART*n*>_Create_UserInit	Performs user-defined initialization relating to the UART interface functions
r_<Config_UART*n*>_interrupt_send	Performs processing in response to the UART communication interrupt
r_<Config_UART*n*>_interrupt_receive	Performs processing in response to the UART reception interrupt
r_<Config_UART*n*>_interrupt_error	Performs processing in response to the UART error interrupt
r_<Config_UART*n*>_callback_sendend	Performs processing in response to the UART communication interrupt
r_<Config_UART*n*>_callback_receiveend	Performs processing in response to the UART reception interrupt
r_<Config_UART*n*>_callback_error	Performs processing in response to the UART error interrupt

R_<Config_UART*n*>_Create

Performs initialization necessary to control the UART interface functions.

Remark This API function is called by R_Systeminit function.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_UART*n*>_Start

Sets UART communication to standby mode.

[Syntax]

```
void    R_<Config_UARTn>_Start(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_UART*n*>_Stop

Ends UART communication.

[Syntax]

```
void    R_<Config_UART n>_Stop(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_UART*n*>_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]

MD_STATUS R_<Config_UART*n*>_Send(uint8_t * const *tx_buf*, uint16_t *tx_num*);

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
I	uint8_t * const *tx_buf*	Pointer to a buffer storing the transmission data
I	uint16_t *tx_num*	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion.
MD_ARGERROR	Invalid argument specification
MD_ERROR	Sending in progress

R_<Config_UART*n*>_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]

MD_STATUS R_<Config_UART*n*>_Receive(uint8_t * const *rx_buf*, uint16_t *rx_num*);

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	uint8_t * const *rx_buf*	Pointer to a buffer to store the received data
I	uint16_t *rx_num*	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_<Config_UART*n*>_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]

```
void R_<Config_UART n>_Create_UserInit(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_UART*n*>_interrupt_send

Performs processing in response to the UART communication interrupt.

[Syntax]

```
void    r_<Config_UART n>_interrupt_send(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_UART*n*>_interrupt_receive

Performs processing in response to the UART reception interrupt.

[Syntax]

```
void    r_<Config_UARTn>_interrupt_receive(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_UART*n*>_interrupt_error

Performs processing in response to the UART error interrupt.

[Syntax]

```
void    r_<Config_UARTn>_interrupt_error(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_UART*n*>_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]

```
void r_<Config_UART n>_callback_sendend(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_UART*n*>_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_UART n>_callback_receiveend(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_UART*n*>_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]

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

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
I	uint32_t *err_type*	Trigger for UART error interrupt 0x00xx01B: Bit error 0x00x10xB: Overrun error 0x001x0xB: Framing error 0100xx0xB: Parity error

[Return value]

None.

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.)

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 */
```

```
volatile uint8_t transmitend_flag_1 = 0;
volatile uint8_t receiveend_flag_1 = 0;
/* End user code. 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 3.16 API Functions: [Watchdog Timer]

API Function Name	Description
R_<Config_WDT*n*>_Create	Performs initialization necessary to control the watchdog timer functions
R_<Config_WDT*n*>_Restart	Clears the watchdog timer counter and resumes counting
R_<Config_WDTX*n*>_Create	Performs initialization necessary to control the watchdog timer functions
R_<Config_WDTX*n*>_Restart	Clears the watchdog timer counter and resumes counting
R_<Config_WDT*n*>_Create_UserInit	Performs user-defined initialization relating to the watchdog timer functions
r_<Config_WDT*n*>_interrupt	Performs processing in response to the interval interrupt
R_<Config_WDTX*n*>_Create_UserInit	Performs user-defined initialization relating to the watchdog timer functions
r_<Config_WDTX*n*>_interrupt	Performs processing in response to the interval interrupt

R_<Config_WDT*n*>_Create

Performs initialization necessary to control the watchdog timer functions.

Remark This API function is called by R_Systeminit function.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_WDT*n*>_Restart

Clears the watchdog timer counter and resumes counting.

[Syntax]

```
void    R_<Config_WDTn_Restart(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_WDT*Xn*>_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_WDT*X*n>_Restart

Clears the watchdog timer counter and resumes counting.

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_WDT*n*>_Create_UserInit

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

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

[Syntax]

```
void R_<Config_WDTn>_Create_UserInit(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_WDT*n*>_interrupt

Performs processing in response to the watchdog timer interrupt.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_WDT*Xn*>_Create_UserInit

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

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

[Syntax]

```
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_WDT*Xn*>_interrupt

Performs processing in response to the watchdog timer interrupt.

[Syntax]

```
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]

API Function Name	Description
R_<Config_TAU*Xn_m*>_Create	Performs initialization necessary to control the clock divide functions
R_<Config_TAU*Xn_m*>_Start	Starts the count for channel *m*
R_<Config_TAU*Xn_m*>_Stop	Ends the count for channel *m*
R_<Config_TAU*Xn_m*>_Create_UserInit	Performs user-defined initialization relating to the clock divide functions
r_<Config_TAU*Xn_m*>_interrupt	Performs processing in response to the timer interrupt
r_<Config_TAU*Xn_m*>_interrupt_pe*k*	Performs processing in response to the timer interrupt or PE*k*

R_<Config_TAU*Xn_m*>_Create

Performs initialization necessary to control the clock divide functions.

Remark This API function is called by R_Systeminit function.

[Syntax]

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

Remark TAU*X* 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*>_Start

Starts the count for channel *m*.

[Syntax]

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

Remark TAU*X* 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]

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

Remark TAU*X* 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*>_*Create*_UserInit

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

Remark This API functions is called as the R_<Config_TAU*Xn_m*>_*Create* callback routine.

[Syntax]

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

Remark TAU*X* 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_pe*k*

Performs processing in response to the timer interrupt for PE*k*.

[Syntax]

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

Remark TAU*X* 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_TAU*Xn_m*>_interrupt

Performs processing in response to the timer interrupt.

[Syntax]

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

Remark TAU*X* 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]

API Function Name	Description
R_<Config_DCRA*n*>_Create	Performs initialization necessary to control the data CRC functions
R_<Config_DCRAn>_InitializeCRCData	Initializes CRC data register with the initial start value
R_<Config_DCRA*n*>_Input32bitData	Sets the calculation data for 32-bit width
R_<Config_DCRA*n*>_Input16bitData	Sets the calculation data for 16-bit width
R_<Config_DCRA*n*>_Input8bitData	Sets the calculation data for 8-bit width
R_<Config_DCRA*n*>_GetResult_32bitData	Reads the results of CRC calculation for 32-bit width
R_<Config_DCRA*n*>_GetResult_16bitData	Reads the results of CRC calculation for 16-bit width
R_<Config_DCRA*n*>_Create_UserInit	Performs user-defined initialization relating to the data CRC functions
R_<Config_KCRC*n*>_Create	Performs initialization necessary to control the data CRC functions
R_<Config_KCRCn>_InitializeCRCData	Initializes CRC data register with the initial start value
R_<Config_KCRC*n*>_Input32bitData	Sets the calculation data for 32-bit width
R_<Config_KCRC*n*>_Input16bitData	Sets the calculation data for 16-bit width
R_<Config_KCRC*n*>_Input8bitData	Sets the calculation data for 8-bit width
R_<Config_KCRC*n*>_GetResult_64bitData	Reads the results of CRC calculation for 64-bit width
R_<Config_KCRC*n*>_GetResult_32bitData	Reads the results of CRC calculation for 32-bit width
R_<Config_KCRC*n*>_GetResult_16bitData	Reads the results of CRC calculation for 16-bit width
R_<Config_KCRC*n*>_GetResult_8bitData	Reads the results of CRC calculation for 8-bit width
R_<Config_KCRC*n*>_Create_UserInit	Performs user-defined initialization relating to the data CRC functions

R_<Config_DCRA*n*>_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_DCRA*n*>_InitializeCRCData

Initializes CRC data register with the initial start value.

[Syntax]

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

Remark *n* is the unit number.

[Argument(s)]

I/O	Argument(s)	Description
I	uint32_t *crc_data*	DCRA initializtion value

[Return value]

None.

R_<Config_DCRA*n*>_Input32bitData

Sets the calculation data for 32-bit width.

[Syntax]

void R_<Config_DCRA*n*>_Input32bitData(const uint32_t * *data*, uint32_t *data_num*);

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
I	const uint32_t * *data*	Pointer to a buffer storing the calculation data
I	uint32_t *data_num*	Total amount of calculation data

[Return value]

None.

R_<Config_DCRA*n*>_Input16bitData

Sets the calculation data for 16-bit width.

[Syntax]

```
void    R_<Config_DCRAn>_Input16bitData(const uint16_t * data, uint32_t data_num);
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
I	const uint16_t * *data*	Pointer to a buffer storing the calculation data
I	uint32_t *data_num*	Total amount of calculation data

[Return value]

None.

R_<Config_DCRA*n*>_Input8bitData

Sets the calculation data for 8-bit width.

[Syntax]

```
void    R_<Config_DCRAn>_Input8bitData(const uint8_t * data, uint32_t data_num);
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
I	const uint8_t * *data*	Pointer to a buffer storing the calculation data
I	uint32_t *data_num*	Total amount of calculation data

[Return value]

None.

R_<Config_DCRA*n*>_GetResult_32bitData

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

[Syntax]

```
void    R_<Config_DCRAn>_GetResult_32bitData(uint32_t * data);
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	uint32_t * *data*	Pointer to area in which to store the results of calculation data

[Return value]

None.

R_<Config_DCRA*n*>_GetResult_16bitData

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

[Syntax]

```
void    R_<Config_DCRAn>_GetResult_16bitData(uint16_t * data);
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	uint16_t * *data*	Pointer to area in which to store the results of calculation data

[Return value]

None.

R_<Config_DCRA*n*>_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]

```
void R_<Config_DCRA n>_Create_UserInit(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_KCRC*n*>_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_KCRC*n*>_InitializeCRCData

Initializes CRC data register with the initial start value.

[Syntax]

void R_<Config_KCRC*n*>_InitializeCRCData(uint32_t *crcout0_data*, uint32_t *crcout1_data*);

Remark *n* is the unit number.

[Argument(s)]

I/O	Argument(s)	Description
I	uint32_t *crcout0_data*	lower side of CRC initialization data
I	uint32_t *crcout1_data*	upper side of CRC initialization data

[Return value]

None.

R_<Config_KCRC*n*>_Input32bitData

Sets the calculation data for 32-bit width.

[Syntax]

void R_<Config_KCRC*n*>_Input32bitData(const uint32_t * *data*, uint32_t *data_num*);

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
I	const uint32_t * *data*	Pointer to a buffer storing the calculation data
I	uint32_t *data_num*	Total amount of calculation data

[Return value]

None.

R_<Config_KCRC*n*>_Input16bitData

Sets the calculation data for 16-bit width.

[Syntax]

```
void    R_<Config_KCRCn>_Input16bitData(const uint16_t * data, uint32_t data_num);
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
I	const uint16_t * *data*	Pointer to a buffer storing the calculation data
I	uint32_t *data_num*	Total amount of calculation data

[Return value]

None.

R_<Config_KCRC*n*>_Input8bitData

Sets the calculation data for 8-bit width.

[Syntax]

```
void    R_<Config_KCRCn>_Input8bitData(const uint8_t * data, uint32_t data_num);
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
I	const uint8_t * *data*	Pointer to a buffer storing the calculation data
I	uint32_t *data_num*	Total amount of calculation data

[Return value]

None.

R_<Config_KCRC*n*>_GetResult_64bitData

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

[Syntax]

```
void    R_<Config_KCRCn>_GetResult_64bitData(uint64_t * data);
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	Uint64_t * *data*	Pointer to area in which to store the results of calculation data

[Return value]

None.

R_<Config_KCRC*n*>_GetResult_32bitData

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

[Syntax]

```
void    R_<Config_KCRCn>_GetResult_32bitData(uint32_t * data);
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	uint32_t * *data*	Pointer to area in which to store the results of calculation data.

[Return value]

None.

R_<Config_KCRC*n*>_GetResult_16bitData

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

[Syntax]

```
void    R_<Config_KCRCn>_GetResult_16bitData(uint16_t * data);
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	uint16_t * *data*	Pointer to area in which to store the results of calculation data

[Return value]

None.

R_<Config_KCRC*n*>_GetResult_8bitData

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

[Syntax]

```
void    R_<Config_KCRCn>_GetResult_8bitData(uint8_t * data);
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	Uint8_t * *data*	Pointer to area in which to store the results of calculation data

[Return value]

None.

R_<Config_KCRC*n*>_Create_UserInit

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

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

[Syntax]

```
void R_<Config_KCRCn>_Create_UserInit(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 3.19 API Functions: [Delay Count]

API Function Name	Description
R_<Config_TAU*Xn_m*>_Create	Performs initialization necessary to control the delay count functions
R_<Config_TAU*Xn_m*>_Start	Starts the count for channel *m*
R_<Config_TAU*Xn_m*>_Stop	Ends the count for channel *m*
R_<Config_TAU*Xn_m*>_Create_UserInit	Performs user-defined initialization relating to the delay count functions
r_<Config_TAU*Xn_m*>_interrupt	Performs processing in response to the timer interrupt
r_<Config_TAU*Xn_m*>_interrupt_pe*k*	Performs processing in response to the timer interrupt for PE*k*

R_<Config_TAU*Xn_m*>_Create

Performs initialization necessary to control the delay count functions.

Remark This API function is called by R_Systeminit function.

[Syntax]

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

Remark TAU*X* 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*>_Start

Starts the count for channel *m*.

[Syntax]

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

Remark TAU*X* 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]

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

Remark TAU*X* 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*>_*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]

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

Remark TAU*X* 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

Performs processing in response to the timer interrupt.

[Syntax]

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

Remark TAU*X* 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_pe*k*

Performs processing in response to the timer interrupt for PE*k*.

[Syntax]

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

Remark TAU*X* 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 3.20 API Functions: [DMA Controller]

API Function Name	Description
R_<Config_DMAC*n*>_Create	Performs initialization necessary to control the DMAC*n* functions
R_DMAC*nm*_Create	Performs initialization necessary to control the DMAC*n* channel*m* functions
R_<Config_DMAC*nm*>_Start	Enables the DMAC*n* channel*m* transfer
R_<Config_DMAC*nm*>_Stop	Disables the DMAC*n* channelm transfer
R_<Config_DMAC*nm*>_Set_SoftwareTrigger	Generates the CMAC*n* channelm transfer request
R_<Config_DMAC*nm*>_Suspend	Suspends DMAC*n* channel*m* transfer
R_<Config_DMAC*nm*>_Resume	Resumes DMAC*n* channel*m* transfer
R_<Config_DMAC*n*>_Create_UserInit	Performs user-defined initialization relating to the DMAC*n* functions
r_<Config_DMAC*nm*>_dmac*nm*_interrupt	Performs processing in response to the DMAC*n* channel*m* interrupt
r_<Config_DMACnm>_Callback_DMAC*nm*_Transfer_Error	Performs processing in response to the DMAC*n* channel*m* transfer error interrupt
R_<Config_SDMAC*nm*>_Create	Performs initialization necessary to control the SDMAC*n* channel*m* functions
R_<Config_SDMAC*nm*>_Start	Enables the SDMAC*n* channel*m* transfer
R_<Config_SDMAC*nm*>_Stop	Disables the SDMAC*n* channel*m* transfer
R_<Config_SDMAC*nm*>_Resume	Resume SDMAC*n* channel*m* transfer
R_<Config_SDMACnm>_Reset	
R_<Config_SDMAC*nm*>_Create_UserInit	Performs user-defined initialization relating to the SDMAC*n* channel*m* functions
r_<Config_SDMAC*nm*>_end_interrupt	Performs processing in response to the SDMAC*n* channel*m* transfer end interrupt
r_<Config_SDMAC*nm*>_Callback_PE*k*_Address_Error	Performs user-defined processing relating to the SDMAC*n* channel*m* address error interrupt for PE*k*
R_<Config_SDMAC*nm*>_Set_DescriptorMemory	Sets descriptor memory for SDMAC*n* channel*m*

R_<Config_DMAC*n*>_Create

Performs initialization necessary to control the DMAC*n* functions

Remark This API function is called by R_Systeminit function.

[Syntax]

```
void R_<Config_DMAC n>_Create(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_DMAC*nm*_Create

Performs initialization necessary to control the DMAC*n* channel*m* functions.

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

[Syntax]

```
void R_DMAC*nm*_Create(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_DMAC*nm*>_Start

Enables the DMAC*n* channel*m* transfer.

[Syntax]

```
void    R_<Config_DMAC nm>_Start(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_DMAC*nm*>_Stop

Disables the DMAC*n* channel*m* transfer.

[Syntax]

```
void    R_<Config_DMAC nm>_Stop(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_DMAC*nm*>_Set_SoftwareTrigger

Generates the DMAC*n* channel*m* transfer request.

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_DMAC*nm*>_Suspend

Suspends DMAC*n* channel*m* transfer.

[Syntax]

```
void    R_<Config_DMAC nm>_Suspend(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_DMAC*nm*>_Resume

Resumes DMAC*n* channel*m* transfer.

[Syntax]

```
void    R_<Config_DMAC nm>_Resume(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_DMAC*n*>_Create_UserInit

Performs user-defined initialization relating to the DMAC*n* functions.

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

[Syntax]

```
void R_<Config_DMACn>_Create_UserInit(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_DMAC*nm*>_dmac*nm*_interrupt

Performs processing in response to the DMAC*n* channel*m* interrupt.

[Syntax]

```
void    r_<Config_DMAC nm>_dmac nm_interrupt(void);
```

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_DMAC*nm*>_Callback_DMAC*nm*_Transfer_Error

Performs processing in response to the DMAC transfer error interrupt.

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

[Syntax]

```
void r_<Config_DMACnm>_Callback_DMACnm_Transfer_Error(void);
```

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_DMAC*nm*>_callback_transfer_completion

Performs processing in response to the DMAC transfer end interrupt.

Remark This API functions is called as the r_<Config_DMAC*nm*>_dmac*nm*_interrupt callback routine.

[Syntax]

```
void r_<Config_DMAC nm>_callback_transfer_completion(void);
```

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_DMAC*nm*>_callback_transfer_count_match

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

Remark This API functions is called as the r_<Config_DMAC*nm*>_dmac*nm*_interrupt callback routine.

[Syntax]

```
void r_<Config_DMAC nm>_callback_transfer_count_match(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_SDMAC*nm*>_Create

Performs initialization necessary to control the SDMAC*n* channel*m* functions

Remark This API function is called by R_Systeminit function.

[Syntax]

```
void R_<Config_SDMAC nm>_Create(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_SDMAC*nm*>_Start

Enables the SDMAC*n* channel *m* transfer.

[Syntax]

```
void    R_<Config_SDMAC nm>_Start(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_SDMAC*nm*>_Stop

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

[Syntax]

```
void    R_<Config_SDMAC nm>_Stop(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_SDMAC*nm*>_Resume

Resumes SDMAC*n* channel *m* transfer.

[Syntax]

```
void    R_<Config_SDMAC nm>_Resume(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_SDMAC*nm*>_Reset

Resets SDMAC*n* channel *m*.

[Syntax]

```
void    R_<Config_SDMAC nm>_Reset(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_SDMAC*nm*>_Create_UserInit

Performs user-defined initialization relating to the SDMAC*n* channel*m* functions.

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

[Syntax]

```
void    R_<Config_SDMAC nm>_Create_UserInit(void);
```

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_SDMAC*nm*>_end_interrupt

Performs processing in response to the SDMAC*n* channel*m* transfer end interrupt.

[Syntax]

```
void    r_<Config_SDMAC nm _interrupt(void);
```

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_SDMAC*nm*>_Callback_PE*k*_Address_Error

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

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

[Syntax]

```
void r_<Config_SDMAC nm>_Callback_PE k_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_SDMAC*nm*>_Set_DescriptorMemory

Sets descriptor memory for DMAC*n* channel *m*.

[Syntax]

```
void R_<Config_SDMAC nm>_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)]

I/O	Argument(s)	Description
I	uint32_t * const address	Descriptor memory address
I	uint32_t * const regValues	Register values
I	uint8_t count_num	Register count.

[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]

API Function Name	Description
R_<Config_TAU*Xn_m*>_Create	Performs initialization necessary to control the external event count functions
R_<Config_TAU*Xn_m*>_Start	Starts the count for channel *m*
R_<Config_TAU*Xn_m*>_Stop	Ends the count for channel *m*
R_<Config_TAU*Xn_m*>_Create_UserInit	Performs user-defined initialization relating to the external event count functions
r_<Config_TAU*Xn_m*>_interrupt	Performs processing in response to the timer interrupt
r_<Config_TAU*Xn_m*>_interrupt_pe*k*	Performs processing in response to the timer interrupt for PE*k*

R_<Config_TAU*Xn_m*>_Create

Performs initialization necessary to control the external event count functions.

Remark This API function is called by R_Systeminit function.

[Syntax]

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

Remark TAU*X* 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*>_Start

Starts the count for channel *m*.

[Syntax]

```
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]

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

Remark TAU*X* 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*>_Create_UserInit

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

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

[Syntax]

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

Remark TAU*X* 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

Performs processing in response to the timer interrupt.

[Syntax]

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

Remark TAU*X* 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_pe*k*

Performs processing in response to the timer interrupt for PE*k*.

[Syntax]

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

Remark TAU*X* 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]

API Function Name	Description
R_<Config_TAU*Xn_m*>_Create	Performs initialization necessary to control the input period count detection functions
R_<Config_TAU*Xn_m*>_Start	Starts the count for channel *m*
R_<Config_TAU*Xn_m*>_Stop	Ends the count for channel *m*
R_<Config_TAU*Xn_m*>_Get_PulseWidth	Reads the input pulse width of the timer
R_<Config_TAU*Xn_m*>_Create_UserInit	Performs user-defined initialization relating to the input period count detection functions
r_<Config_TAU*Xn_m*>_interrupt	Performs processing in response to the timer interrupt
r_<Config_TAU*Xn_m*>_interrupt_pe*k*	Performs processing in response to the timer interrupt for PE*k*

R_<Config_TAU*Xn_m*>_Create

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

Remark This API function is called by R_Systeminit function.

[Syntax]

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

Remark TAU*X* 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*>_Start

Starts the count for channel *m*.

[Syntax]

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

Remark TAU*X* 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]

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

Remark TAU*X* 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]

void R_<Config_TAU*Xn_m*>_Get_PulseWidth(uint32_t * const *width*);

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

[Argument(s)]

I/O	Argument(s)	Description
O	uint32_t * const *width*	Pointer to area in which to store the results of input pulse width

[Return value]

None.

R_<Config_TAU*Xn_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_TAU*Xn_m*>_Create callback routine.

[Syntax]

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

Remark TAU*X* 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

Performs processing in response to the timer interrupt.

[Syntax]

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

Remark TAU*X* 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_pe*k*

Performs processing in response to the timer interrupt for PE*k*.

[Syntax]

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

Remark TAU*X* 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]

API Function Name	Description
R_<Config_TAU*Xn_m*>_Create	Performs initialization necessary to control the input position detection functions
R_<Config_TAU*Xn_m*>_Start	Starts the count for channel *m*
R_<Config_TAU*Xn_m*>_Stop	Ends the count for channel *m*
R_<Config_TAU*Xn_m*>_Get_PulseWidth	Reads the input pulse width of the timer
R_<Config_TAU*Xn_m*>_Create_UserInit	Performs user-defined initialization relating to the input position detection functions
r_<Config_TAU*Xn_m*>_interrupt	Performs processing in response to the timer interrupt
r_<Config_TAU*Xn_m*>_interrupt_pe*k*	Performs processing in response to the timer interrupt for PE*k*

R_<Config_TAU*Xn_m*>_Create

Performs initialization necessary to control the input position detection functions.

Remark This API function is called by R_Systeminit function.

[Syntax]

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

Remark TAU*X* 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*>_Start

Starts the count for channel *m*.

[Syntax]

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

Remark TAU*X* 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]

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

Remark TAU*X* 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]

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

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

[Argument(s)]

I/O	Argument(s)	Description
O	uint32_t * const *width*	Pointer to area in which to store the results of input pulse width

[Return value]

None.

R_<Config_TAU*Xn_m*>_Create_UserInit

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

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

[Syntax]

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

Remark TAU*X* 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

Performs processing in response to the timer interrupt.

[Syntax]

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

Remark TAU*X* 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_pe*k*

Performs processing in response to the timer interrupt for PE*k*.

[Syntax]

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

Remark TAU*X* 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.

3.2.24 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]

API Function Name	Description
R_<Config_TAU*Xn_m*>_Create	Performs initialization necessary to control the input pulse interval judgment functions.
R_<Config_TAU*Xn_m*>_Start	Starts the count for channel *m*.
R_<Config_TAU*Xn_m*>_Stop	Ends the count for channel *m*.
R_<Config_TAU*Xn_m*>_Create_UserInit	Performs user-defined initialization relating to the input pulse interval judgment functions.
r_<Config_TAU*Xn_m*>_interrupt	Performs processing in response to the timer interrupt.
r_<Config_TAU*Xn_m*>_interrupt_pe*k*	Performs processing in response to the timer interrupt for PE*k*

R_<Config_TAU*Xn_m*>_Create

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

Remark This API function is called by R_Systeminit function.

[Syntax]

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

Remark TAU*X* 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*>_Start

Starts the count for channel *m*.

[Syntax]

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

Remark TAU*X* 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]

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

Remark TAU*X* 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*>_Create_UserInit

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

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

[Syntax]

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

Remark TAU*X* 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

Performs processing in response to the timer interrupt.

[Syntax]

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

Remark TAU*X* 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_pe*k*

Performs processing in response to the timer interrupt for PE*k*.

[Syntax]

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

Remark TAU*X* 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]

API Function Name	Description
R_<Config_TAU*Xn_m*>_Create	Performs initialization necessary to control the input signal width judgment functions
R_<Config_TAU*Xn_m*>_Start	Starts the count for channel *m*
R_<Config_TAU*Xn_m*>_Stop	Ends the count for channel *m*
R_<Config_TAU*Xn_m*>_Create_UserInit	Performs user-defined initialization relating to the input signal width judgment functions
r_<Config_TAU*Xn_m*>_interrupt	Performs processing in response to the timer interrupt
r_<Config_TAU*Xn_m*>_interrupt_pe*k*	Performs processing in response to the timer interrupt for PE*k*

R_<Config_TAU*Xn_m*>_Create

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

Remark This API function is called by R_Systeminit function.

[Syntax]

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

Remark TAU*X* 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*>_Start

Starts the count for channel *m*.

[Syntax]

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

Remark TAU*X* 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]

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

Remark TAU*X* 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*>_Create_UserInit

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

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

[Syntax]

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

Remark TAU*X* 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]

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

Remark TAU*X* 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_pe*k*

Performs processing in response to the timer interrupt for PE*k*.

[Syntax]

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

Remark TAU*X* 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]

API Function Name	Description
R_<Config_TAU*Xn_m*>_Create	Performs initialization necessary to control the input signal width measurement functions.
R_<Config_TAU*Xn_m*>_Start	Starts the count for channel *m*.
R_<Config_TAU*Xn_m*>_Stop	Ends the count for channel *m*.
R_<Config_TAU*Xn_m*>_Get_PulseWidth	Reads the input pulse width of the timer.
R_<Config_TAU*Xn_m*>_Create_UserInit	Performs user-defined initialization relating to the input signal width measurement functions.
r_<Config_TAU*Xn_m*>_interrupt	Performs processing in response to the timer interrupt.
r_<Config_TAU*Xn_m*>_interrupt_pe*k*	Performs processing in response to the timer interrupt for PE*k*.

R_<Config_TAU*Xn_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*X* 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*>_Start

Starts the count for channel *m*.

[Syntax]

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

Remark TAU*X* 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]

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

Remark TAU*X* 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]

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

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

[Argument(s)]

I/O	Argument(s)	Description
O	uint32_t * const *width*	Pointer to area in which to store the results of input pulse width

[Return value]

None.

R_<Config_TAU*Xn_m*>_Create_UserInit

Performs user-defined initialization relating to the input signal width measurement 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 TAU*X* 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

Performs processing in response to the timer interrupt.

[Syntax]

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

Remark TAU*X* 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_pe*k*

Performs processing in response to the timer interrupt for PE*k*.

[Syntax]

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

Remark TAU*X* 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]

API Function Name	Description
R_<Config_KEY>_Create	Performs initialization necessary to control the key interrupt functions
R_<Config_KEY>_Start	Enables the acceptance of the key interrupt
R_<Config_KEY>_Stop	Disables the acceptance of the key interrupt
R_<Config_KEY>_Create_UserInit	Performs user-defined initialization relating to the key interrupt functions
r_<Config_KEY>_interrupt	Performs processing in response to the key interrupt

R_<Config_KEY>_Create

Performs initialization necessary to control the key interrupt functions.

Remark This API function is called by R_Systeminit function.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_KEY>_Start

Enables the acceptance of the key interrupt.

[Syntax]

```
void    R_<Config_KEY>_Start(void);
```

[Argument(s)]

None.

[Return value]

None.

R_<Config_KEY>_Stop

Disables the acceptance of the key interrupt.

[Syntax]

```
void    R_<Config_KEY>_Stop(void);
```

[Argument(s)]

None.

[Return value]

None.

R_<Config_KEY>_Create_UserInit

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

Remark This API 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]

```
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]

API Function Name	Description
R_<Config_TAU*Xn_m*>_Create	Performs initialization necessary to control the one pulse output functions.
R_<Config_TAU*Xn_m*>_Start	Starts the count for channel *m*.
R_<Config_TAU*Xn_m*>_Stop	Ends the count for channel *m*.
R_<Config_TAU*Xn_m*>_Create_UserInit	Performs user-defined initialization relating to the one pulse output functions.
r_<Config_TAU*Xn_m*>_interrupt	Performs processing in response to the timer interrupt.
r_<Config_TAU*Xn_m*>_interrupt_pe*k*	Performs processing in response to the timer interrupt for PE*k*.

R_<Config_TAU*Xn_m*>_Create

Performs initialization necessary to control the one pulse output functions.

Remark This API function is called by R_Systeminit function.

[Syntax]

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

Remark TAU*X* 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*>_Start

Starts the count for channel *m*.

[Syntax]

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

Remark TAU*X* 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]

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

Remark TAU*X* 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*>_Create_UserInit

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

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

[Syntax]

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

Remark TAU*X* 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

Performs processing in response to the timer interrupt.

[Syntax]

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

Remark TAU*X* 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_pe*k*

Performs processing in response to the timer interrupt for PE*k*.

[Syntax]

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

Remark TAU*X* 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.

3.2.29 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]

API Function Name	Description
R_<Config_TAU*Xn*>_Create	Performs initialization necessary to control the one-shot pulse output functions
R_<Config_TAU*Xn*>_Start	Starts the count
R_<Config_TAU*Xn*>_Stop	Ends the count
R_<Config_TAU*Xn*>_SoftwareTriggerOn	Generates the TAU channel start trigger by software
R_<Config_TAU*Xn*>_Create_UserInit	Performs user-defined initialization relating to the one-shot pulse output functions
r_<Config_TAU*Xn*>_channel*m*_interrupt	Performs processing in response to the timer interrupt
r_<Config_TAU*Xn*>_channel*m*_interrupt_pek	Performs processing in response to the timer interrupt for PE*k*

R_<Config_TAU*Xn*>_Create

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

Remark This API function is called by R_Systeminit function.

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_TAU*Xn*>_Start

Starts the count.

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_TAU*Xn*>_Stop

Ends the count.

[Syntax]

```
void    R_<Config_TAUXn>_Stop(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_TAU*Xn*>_SoftwareTriggerOn

Generates the TAU channel start trigger by software.

[Syntax]

```
void    R_<Config_TAUXn>_SoftwareTriggerOn(void);
```

Remark TAU*X* 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 functions is called as the R_<Config_TAUXn>_Create callback routine.

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_TAU*Xn*>_channel*m*_interrupt_pe*k*

Performs processing in response to the timer interrupt for PE*k*

[Syntax]

```
void    r_<Config_TAUXn>_channelm _interrupt_pek(void);
```

Remark TAU*X* 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_TAU*Xn*>_channel*m*_interrupt

Performs processing in response to the timer interrupt.

[Syntax]

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

Remark TAU*X* 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 3.30 API Functions: [Overflow Interrupt Output (input period count detecting)]

API Function Name	Description
R_<Config_TAU*Xn_m*>_Create	Performs initialization necessary to control the overflow interrupt output (input period count detecting) functions
R_<Config_TAU*Xn_m*>_Start	Starts the count for channel *m*
R_<Config_TAU*Xn_m*>_Stop	Ends the count for channel *m*
R_<Config_TAU*Xn_m*>_Create_UserInit	Performs user-defined initialization relating to the overflow interrupt output (input period count detecting) functions
r_<Config_TAU*Xn_m*>_interrupt	Performs processing in response to the timer interrupt
r_<Config_TAU*Xn_m*>_interrupt_pe*k*	Performs processing in response to the timer interrupt for PE*k*

R_<Config_TAU*Xn_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]

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

Remark TAU*X* 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*>_Start

Starts the count for channel *m*.

[Syntax]

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

Remark TAU*X* 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]

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

Remark TAU*X* 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*>_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_TAU*Xn_m*>_Create callback routine.

[Syntax]

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

Remark TAU*X* 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

Performs processing in response to the timer interrupt.

[Syntax]

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

Remark TAU*X* 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_pe*k*

Performs processing in response to the timer interrupt for PE*k*.

[Syntax]

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

Remark TAU*X* 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)]

API Function Name	Description
R_<Config_TAU*Xn_m*>_Create	Performs initialization necessary to control the overflow interrupt output (width measurement) functions
R_<Config_TAU*Xn_m*>_Start	Starts the count for channel *m*
R_<Config_TAU*Xn_m*>_Stop	Ends the count for channel *m*
R_<Config_TAU*Xn_m*>_Get_PulseWidth	Reads the input pulse width of the timer
R_<Config_TAU*Xn_m*>_Create_UserInit	Performs user-defined initialization relating to the overflow interrupt output (width measurement) functions
r_<Config_TAU*Xn_m*>_interrupt	Performs processing in response to the timer interrupt
r_<Config_TAU*Xn_m*>_interrupt_pe*k*	Performs processing in response to the timer interrupt for PE*k*

R_<Config_TAU*Xn_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]

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

Remark TAU*X* 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*>_Start

Starts the count for channel *m*.

[Syntax]

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

Remark TAU*X* 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]

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

Remark TAU*X* 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*>_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_TAU*Xn_m*>_Create callback routine.

[Syntax]

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

Remark TAU*X* 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

Performs processing in response to the timer interrupt.

[Syntax]

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

Remark TAU*X* 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_pe*k*

Performs processing in response to the timer interrupt for PE*k*.

[Syntax]

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

Remark TAU*X* 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.

3.2.32 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]

API Function Name	Description
R_<Config_RIIC*n*>_Create	Performs initialization necessary to control the IIC master mode functions
R_<Config_RIIC*n*>_Start	Enables IIC
R_<Config_RIIC*n*>_Stop	Disables IIC
R_<Config_RIIC*n*>_Master_Send	Starts master transmission
R_<Config_RIIC*n*>_Master_Send_Without_Stop	Starts master transmission (with no stop condition being issued at the end of transmission)
R_<Config_RIIC*n*>_Master_Receive	Starts master reception
R_<Config_RIIC*n*>_StartCondition	Issues a start condition
R_<Config_RIIC*n*>_StopCondition	Issues a stop condition
R_<Config_RIIC*n*>_Create_UserInit	Performs user-defined initialization relating to the IIC master mode functions
r_<Config_RIIC*n*>_error_interrupt	Executes processing in response to communication error interrupts or communication event generation interrupts
r_<Config_RIIC*n*>_receive_interrupt	Executes processing in response to receive data full interrupts
r_<Config_RIIC*n*>_transmit_interrupt	Executes processing in response to transmit data empty interrupts
r_<Config_RIIC*n*>_transmitend_interrupt	Executes processing in response to transmit end interrupts
r_<Config_RIIC*n*>_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. In master transmission without issuing a stop condition, this API function executes processing in response to transmit end interrupts
r_<Config_RIIC*n*>_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
r_<Config_RIIC*n*>_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

R_<Config_RIIC*n*>_Create

Performs initialization necessary to control the IIC master mode functions.

Remark This API function is called by R_Systeminit function.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_RIIC*n*>_Start

Enables IIC.

[Syntax]

```
void    R_<Config_RIICn>_Start(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_RIIC*n*>_Stop

Disables IIC.

[Syntax]

```
void    R_<Config_RIICn>_Stop(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_RIIC*n*>_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_RIIC*n*>_StartCondition to start master transmission.

Remark3 A stop condition is issued in r_<Config_RIIC*n*>_transmitend_interrupt to stop master transmission.

Remark4 Calling R_<Config_RIIC*n*>_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)]

I/O	Argument(s)	Description
I	uint16_t *adr*	Slave address
I	uint8_t * const *tx_buf*	Pointer to the buffer where the data to be transmitted are stored
I	uint16_t *tx_num*	Number of bytes to be transmitted

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	The bus is busy
MD_ERROR2	The specification of argument *adr* is invalid

R_<Config_RIIC*n*>_Master_Send_Without_Stop

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

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

Remark2 This API function internally calls R_<Config_RIIC*n*>_StartCondition to start master transmission.

Remark3 r_<Config_RIIC*n*>_transmitend_interrupt does not issue a stop condition to stop maste transmission.

Remark4 Calling R_<Config_RIIC*n*>_Start is required before this API function is called to execute master transmission.

[Syntax]

MD_STATUS R_<Config_RIICn>_Master_Send_Without_Stop(uint16_t *adr*, uint8_t * const *tx_buf*, uint16_t *tx_num*);

[Argument(s)]

I/O	Argument(s)	Description
I	uint16_t *adr*	Slave address
I	uint8_t * const *tx_buf*	Pointer to the buffer where the data to be transmitted are stored
I	uint16_t *tx_num*	Number of bytes to be transmitted

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	The bus is busy
MD_ERROR2	The specification of argument *adr* is invalid

R_<Config_RIIC*n*>_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_RIIC*n*>_StartCondition to start master reception.

Remark3 A stop condition is issued in r_<Config_RIIC*n*>_receive_interrupt to stop master reception.

Remark4 Calling R_<Config_RIIC*n*>_Start is required before this API function is called to execute master reception.

[Syntax]

MD_STATUS R_<Config_RIIC*n*>_Master_Receive(uint16_t *adr*, uint8_t * const *rx_buf*, uint16_t *rx_num*);

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
I	uint16_t *adr*	Slave address
O	uint8_t * const *rx_buf*	Pointer to the buffer where the received data are to be stored
I	uint16_t *rx_num*	Number of bytes to be received

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	The bus is busy
MD_ERROR2	The specification of argument *adr* is invalid

R_<Config_RIIC*n*>_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_RIIC*n*>_error_interrupt is called.

[Syntax]

```
void    R_<Config_RIICn>_StartCondition(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_RIIC*n*>_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_RIIC*n*>_error_interrupt is called.

[Syntax]

```
void    R_<Config_RIICn>_StopCondition(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_RIIC*n*>_Create_UserInit

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

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

[Syntax]

```
void R_<Config_RIICn>_Create_UserInit(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_RIIC*n*>_error_interrupt

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

[Syntax]

```
void    r_<Config_RIICn>_error_interrupt(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_RIIC*n*>_receive_interrupt

Executes processing in response to receive data full interrupts.

[Syntax]

```
void    r_<Config_RIICn>_receive_interrupt(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_RIIC*n*>_transmit_interrupt

Executes processing in response to transmit data empty interrupts.

[Syntax]

```
void    r_<Config_RIICn>_transmit_interrupt(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_RIIC*n*>_transmitend_interrupt

Executes processing in response to transmit end interrupts.

[Syntax]

```
void    r_<Config_RIICn>_transmitend_interrupt(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_RIIC*n*>_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_RIIC*n*>_error_interrupt as a callback routine.

Remark2 To execute master transmission, call R_<Config_RIIC*n*>_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_RIIC*n*>>_transmitend_interrupt as a callback routine.

Remark4 To execute master transmission, call R_<Config_RIIC*n*>_Master_Send_Without_Stop.

[Syntax]

```
void    r_<Config_RIICn>_callback_transmitend(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_RIIC*n*>_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.

Remark1 This API function is called by r_<Config_RIICn>_error_interrupt as a callback routine.

Remark2 To execute master reception, call R_<Config_RIICn>_Master_Receive.

[Syntax]

```
void    r_<Config_RIIC n>_callback_receiveend(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_RIIC*n*>_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]

```
void r_<Config_RIICn>_callback_receiveerror(MD_STATUS status);
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
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};
uint8_t rx_buf[6] = {0};
volatile uint8_t riic0_master_sendend_flag = 0x00U;
volatile uint8_t riic1_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(riic0_master_sendend_flag == 0 || riic0_slave_reveiveend_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 riic0_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 */
    riic0_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 riic1_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 */
    riic1_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]

API Function Name	Description
R_<Config_RIIC*n*>_Create	Performs initialization necessary to control the IIC slave mode functions
R_<Config_RIIC*n*>_Start	Enables IIC
R_<Config_RIIC*n*>_Stop	Disables IIC
R_<Config_RIIC*n*>_Slave_Send	Starts slave transmission
R_<Config_RIIC*n*>_Slave_Receive	Starts slave reception
R_<Config_RIIC*n*>_StartCondition	Issues a start condition
R_<Config_RIIC*n*>_StopCondition	Issues a stop condition
R_<Config_RIIC*n*>_Create_UserInit	Performs user-defined initialization relating to the IIC slave mode functions
r_<Config_RIIC*n*>_error_interrupt	Executes processing in response to communication error interrupts or communication event generation interrupts
r_<Config_RIIC*n*>_receive_interrupt	Executes processing in response to receive data full interrupts
r_<Config_RIIC*n*>_transmit_interrupt	Executes processing in response to transmit data empty interrupts
r_<Config_RIIC*n*>_transmitend_interrupt	Executes processing in response to transmit end interrupts
r_<Config_RIIC*n*>_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
r_<Config_RIIC*n*>_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 interrupt
r_<Config_RIIC*n*>_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

R_<Config_RIIC*n*>_Create

Performs initialization necessary to control the IIC slave mode functions.

Remark This API function is called by R_Systeminit function.

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_RIIC*n*>_Start

Enables IIC.

[Syntax]

```
void    R_<Config_RIICn>_Start(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_RIIC*n*>_Stop

Disables IIC.

[Syntax]

```
void    R_<Config_RIICn>_Stop(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_RIIC*n*>_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]

```
MD_STATUS    R_<Config_RIICn>_Slave_Send(uint8_t * const tx_buf, uint16_t tx_num);
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
I	uint8_t * const *tx_buf*	Pointer to the buffer where the data to be transmitted are stored
I	uint16_t *tx_num*	Number of bytes to be transmitted

[Return value]

Macro	Description
MD_OK	Normal completion

R_<Config_RIIC*n*>_Slave_Receive

Starts slave reception.

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

Remark2 Calling R_<Config_RIICn>_Start is required before this API function is called to execute slave reception.

[Syntax]

```
MD_STATUS    R_<Config_RIICn>_Slave_Receive(uint8_t * const rx_buf, uint16_t rx_num);
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	uint8_t * const *rx_buf*	Pointer to the buffer where the received data are to be stored
I	uint16_t *rx_num*	Number of bytes to be received

[Return value]

Macro	Description
MD_OK	Normal completion

R_<Config_RIIC*n*>_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]

```
void    R_<Config_RIICn>_StartCondition(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_RIIC*n*>_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]

```
void    R_<Config_RIIC n>_StopCondition(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_RIIC*n*>_Create_UserInit

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

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

[Syntax]

```
void R_<Config_RIIC n>_Create_UserInit(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_RIIC*n*>_error_interrupt

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

[Syntax]

```
void    r_<Config_RIICn>_error_interrupt(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_RIIC*n*>_receive_interrupt

Executes processing in response to receive data full interrupts.

[Syntax]

```
void    r_<Config_RIICn>_receive_interrupt(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_RIIC*n*>_transmit_interrupt

Executes processing in response to transmit data empty interrupts.

[Syntax]

```
void    r_<Config_RIICn>_transmit_interrupt(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_RIIC*n*>_transmitend_interrupt

Executes processing in response to transmit end interrupts.

[Syntax]

```
void    r_<Config_RIICn>_transmitend_interrupt(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_RIIC*n*>_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]

```
void    r_<Config_RIICn>_callback_transmitend(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_RIIC*n*>_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]

```
void    r_<Config_RIICn>_callback_receiveend(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_RIIC*n*>_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]

```
void r_<Config_RIICn>_callback_receiveerror(MD_STATUS status);
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
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

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 3.34 API Functions: [SCI3 Clock Synchronous Mode]

API Function Name	Description
R_<Config_SCI3*n*>_Create	Performs initialization necessary to control the SCI3*n* Clock Synchronous Mode functions
R_<Config_SCI3*n*>_Start	Enables SCI3*n*
R_<Config_SCI3*n*>_Stop	Disables SCI3*n*
R_<Config_SCI3*n*>_Send	Starts SCI3*n* data transmission
R_<Config_SCI3*n*>_Receive	Starts SCI3*n* data reception
R_<Config_SCI3*n*>_Create_UserInit	Performs user-defined initialization relating to the SCI3*n* Clock Synchronous Mode functions
r_<Config_SCI3*n*>_interrupt_send	Performs processing in response to the SCI3*n* Clock Synchronous Mode communication interrupt
r_<Config_SCI3*n*>_interrupt_receive	Performs processing in response to the SCI3*n* reception interrupt
r_<Config_SCI3*n*>_interrupt_error	Performs processing in response to the SCI3*n* error interrupt
r_<Config_SCI3*n*>_interrupt_sendend	Performs processing in response to the SCI3*n* send end interrupt
r_<Config_SCI3*n*>_callback_sendend	Performs processing in response to the SCI3*n* communication interrupt
r_<Config_SCI3*n*>_callback_receiveend	Performs processing in response to the SCI3*n* reception interrupt
r_<Config_SCI3*n*>_callback_error	Performs processing in response to the SCI3n error interrupt

R_<Config_SCI3*n*>_Create

Performs initialization necessary to control the SCI3*n* Clock Synchronous Mode functions.

Remark This API function is called by R_Systeminit function.

[Syntax]

```
void R_<Config_SCI3n>_Create(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_SCI3*n*>_Start

Enables SCI3*n*.

[Syntax]

```
void    R_<Config_SCI3n>_Start(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_SCI3*n*>_Stop

Disables SCI3*n*.

[Syntax]

```
void    R_<Config_SCI3n>_Stop(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_SCI3*n*>_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_SCI3*n*>_Start must be called before this API function is called.

[Syntax]

MD_STATUS R_<Config_SCI3*n*>_Send (uint8_t * const *tx_buf*, uint16_t *tx_num*);

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
I	uint8_t * const *tx_buf*	Pointer to a buffer storing the transmission data
I	uint16_t *tx_num*	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_<Config_SCI3*n*>_Receive

Starts SCI3*n* 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_SCI3*n*>_Start is then called.

[Syntax]

```
MD_STATUS  R_<Config_SCI3n>_Receive(uint8_t * const rx_buf, uint16_t rx_num);
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	uint8_t * const *rx_buf*	Pointer to a buffer to store the received data
I	uint16_t *rx_num*	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_<Config_SCI3*n*>_Create_UserInit

Performs user-defined initialization relating to the SCI3*n* Clock Synchronous Mode functions.

Remark This API functions is called as the R_<Config_SCI3*n*>_Create callback routine.

[Syntax]

```
void R_<Config_SCI3n>_Create_UserInit(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_SCI3*n*>_interrupt_send

Performs processing in response to the SCI3*n* communication interrupt.

[Syntax]

```
void    r_<Config_SCI3n>_interrupt_send(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_SCI3*n*>_interrupt_receive

Performs processing in response to the SCI3*n* reception interrupt.

[Syntax]

```
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]

```
void    r_<Config_SCI3n>_interrupt_error(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_SCI3*n*>_interrupt_sendend

Performs processing in response to the SCI3*n* send end interrupt.

[Syntax]

```
void    r_<Config_SCI3n>_interrupt_sendend(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_SCI3*n*>_callback_sendend

Performs processing in response to the SCI3*n* communication interrupt.

Remark This API function is called as the callback routine of interrupt process r_<Config_SCI3*n*>_interrupt_send corresponding to the SCI3 communication interrupt.

[Syntax]

```
void r_<Config_SCI3n>_callback_sendend(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_SCI3*n*>_callback_receiveend

Performs processing in response to the SCI3*n* reception interrupt.

Remark This API function is called as the callback routine of interrupt process r_<Config_SCI3*n*>_interrupt_receive corresponding to the SCI3 error interrupt.

[Syntax]

```
void r_<Config_SCI3n>_callback_receiveend(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_SCI3*n*>_callback_error

Performs processing in response to the SCI3*n* error interrupt.

Remark This API function is called as the callback routine of interrupt process r_<Config_SCI3*n*>_interrupt_error corresponding to the SCI3 error interrupt.

[Syntax]

```
void r_<Config_SCI3n>_callback_error(uint32_t err_type);
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
I	uint32_t *err_type*	Trigger for SCI3 error interrupt 0xxxxxxxB: Transmit Data Register Empty x1xxxxxxB: Receive Data Register Full xx1xxxxxB: Overrun Error xxx1xxxxB: Framing Error xxxx1xxxB: Parity Error

[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.)

r_cg_main.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);
    receiveend_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);
    receiveend_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]

API Function Name	Description
R_<Config_SCI3*n*>_Create	Performs initialization necessary to control the SCI3*n* Asynchronous Mode functions
R_<Config_SCI3*n*>_Start	Enables SCI3*n*
R_<Config_SCI3*n*>_Stop	Disables SCI3*n*
R_<Config_SCI3*n*>_Send	Starts SCI3*n* data transmission
R_<Config_SCI3*n*>_Receive	Starts SCI3*n* data reception
R_<Config_SCI3*n*>_Multiprocessor_Send	Starts SCI3*n* multiprocessor data transmission
R_<Config_SCI3*n*>_Multiprocessor_Receive	Starts SCI3*n* multiprocessor data reception
R_<Config_SCI3*n*>_Create_UserInit	Performs user-defined initialization relating to the SCI3*n* Asynchronous Mode functions
r_<Config_SCI3*n*>_interrupt_send	Performs processing in response to the SCI3*n* Asynchronous Mode communication interrupt
r_<Config_SCI3*n*>_interrupt_receive	Performs processing in response to the SCI3*n* reception interrupt
r_<Config_SCI3*n*>_interrupt_error	Performs processing in response to the SCI3*n* error interrupt
r_<Config_SCI3*n*>_interrupt_sendend	Performs processing in response to the SCI3*n* send end interrupt
r_<Config_SCI3*n*>_callback_sendend	Performs processing in response to the SCI3*n* communication interrupt
r_<Config_SCI3*n*>_callback_receiveend	Performs processing in response to the SCI3*n* reception interrupt
r_<Config_SCI3*n*>_callback_error	Performs processing in response to the SCI3*n* error interrupt

R_<Config_SCI3*n*>_Create

Performs initialization necessary to control the SCI3*n* Asynchronous Mode functions.

Remark This API function is called by R_Systeminit function.

[Syntax]

```
void R_<Config_SCI3n>_Create(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_SCI3*n*>_Start

Enables SCI3*n*.

[Syntax]

```
void    R_<Config_SCI3n>_Start(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_SCI3*n*>_Stop

Disables SCI3*n*.

[Syntax]

```
void    R_<Config_SCI3n>_Stop(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_SCI3*n*>_Send

Starts SCI3*n* 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_SCI3*n*>_Start must be called before this API function is called.

[Syntax]

MD_STATUS R_<Config_SCI3*n*>_Send (uint8_t * const *tx_buf*, uint16_t *tx_num*);

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
I	uint8_t * const *tx_buf*	Pointer to a buffer storing the transmission data
I	uint16_t *tx_num*	Total amount of data to send

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_<Config_SCI3*n*>_Receive

Starts SCI3*n* 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_SCI3*n*>_Start is then called.

[Syntax]

```
MD_STATUS   R_<Config_SCI3n>_Receive(uint8_t * const rx_buf, uint16_t rx_num);
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	uint8_t * const *rx_buf*	Pointer to a buffer to store the received data
I	uint16_t *rx_num*	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_<Config_SCI3*n*>_Multiprocessor_Send

Starts SCI3*n* 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_SCI3*n*>_Start must be called before this API function is called.

[Syntax]

MD_STATUS R_<Config_SCI3*n*>_Multiprocessor_Send (uint8_t * const *tx_buf*, uint16_t *tx_num*, uint8_t *tx_id*);

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
I	uint8_t * const *tx_buf*	Pointer to a buffer storing the transmission data
I	uint16_t *tx_num*	Total amount of data to send
I	uint16_t *tx_id*	Transmit unique ID code

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_<Config_SCI3*n*>_Multiprocessor_Receive

Starts SCI3*n* 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_SCI3*n*>_Start is then called.

[Syntax]

MD_STATUS R_<Config_SCI3*n*>_Multiprocessor_Receive(uint8_t * const *rx_buf*, uint16_t *rx_num*, uint8_t *rx_id*);

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
O	uint8_t * const *rx_buf*	Pointer to a buffer to store the received data
I	uint16_t *rx_num*	Total amount of data to receive
I	uint16_t *rx_id*	Receive unique ID code

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_<Config_SCI3*n*>_Create_UserInit

Performs user-defined initialization relating to the SCI3n Asynchronous Mode functions.

Remark This API functions is called as the R_<Config_SCI3*n*>_Create callback routine.

[Syntax]

```
void R_<Config_SCI3n>_Create_UserInit(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_SCI3*n*>_interrupt_send

Performs processing in response to the SCI3*n* communication interrupt.

[Syntax]

```
void    r_<Config_SCI3n>_interrupt_send(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_SCI3*n*>_interrupt_receive

Performs processing in response to the SCI3*n* reception interrupt.

[Syntax]

```
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]

```
void    r_<Config_SCI3n>_interrupt_error(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_SCI3*n*>_interrupt_sendend

Performs processing in response to the SCI3*n* send end interrupt.

[Syntax]

```
void    r_<Config_SCI3n>_interrupt_sendend(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_SCI3*n*>_callback_sendend

Performs processing in response to the SCI3*n* communication interrupt.

Remark This API function is called as the callback routine of interrupt process r_<Config_SCI3*n*>_interrupt_send corresponding to the SCI3 communication interrupt.

[Syntax]

```
void    r_<Config_SCI3n>_callback_sendend(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_SCI3*n*>_callback_receiveend

Performs processing in response to the SCI3*n* reception interrupt.

Remark This API function is called as the callback routine of interrupt process r_<Config_SCI3*n*>_interrupt_receive corresponding to the SCI3 error interrupt.

[Syntax]

```
void r_<Config_SCI3n>_callback_receiveend(void);
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_SCI3*n*>_callback_error

Performs processing in response to the SCI3*n* error interrupt.

Remark This API function is called as the callback routine of interrupt process r_<Config_SCI3*n*>_interrupt_error corresponding to the SCI3 error interrupt.

[Syntax]

```
void r_<Config_SCI3n>_callback_error(uint32_t err_type);
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument(s)	Description
I	uint32_t *err_type*	Trigger for SCI3 error interrupt 0xxxxxxxB: Transmit Data Register Empty x1xxxxxxB: Receive Data Register Full xx1xxxxxB: Overrun Error xxx1xxxxB: Framing Error xxxx1xxxB: Parity Error

[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

```
/* 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

```
/* 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]

API Function Name	Description
R_<Config_MSPI*nm*>_Create	Performs initialization necessary to control the MSPI Master functions
R_<Config_MSPI*nm*>_Start	Enables MSPI*n*
R_<Config_MSPI*nm*>_Stop	Disables MSPI*n*
R_<Config_MSPI*nm*>_Send	Starts MSPI*n* data transmission
R_<Config_MSPI*nm*>_Receive	Starts MSPI*n* data reception
R_<Config_MSPI*nm*>_Software_Trigger	Sets channel enable and start trigger
R_<Config_MSPI*nm*>_Create_UserInit	Performs user-defined initialization relating to the MSPI Master functions
r_<Config_MSPI*nm*>_channelnm_interrupt_send	Channel send interrupt
r_<Config_MSPI*nm*>_channelnm_interrupt_receive	Channel receive interrupt
r_<Config_MSPI*nm*>_Callback_Interrupt_Send	MSPI*n* channel*m* responses communication interrupt
r_<Config_MSPI*nm*>_Callback_Interrupt_Receive	MSPI*n* channel*m* responses reception interrupt
r_<Config_MSPI*nm*>_Callback_Interrupt_error	MSPI*n* channel*m* responses error interrupt
r_<Config_MSPI*nm*>_Callback_Interrupt_Frameend	Performs processing in response to the MSPI*nm* frameend interrupt
r_<Config_MSPI*nm*>_callback_sendend	Performs processing in response to the MSPI*nm* communication interrupt
r_<Config_MSPI*nm*>_callback_receiveend	Performs processing in response to the MSPI*nm* reception interrupt
r_<Config_MSPI*nm*>_callback_error	Performs processing in response to the MSPI*nm* error interrupt

R_<Config_MSPI*nm*>_Create

Performs initialization necessary to control the MSPI Master functions.

Remark This API function is called by R_Systeminit function.

[Syntax]

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

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_MSPI*nm*>_Start

Enables MSPI*n*.

[Syntax]

```
void    R_<Config_MSPInm>_Start(void);
```

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]

```
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_MSPI*nm*>_Send

Starts MSPI*nm* 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_MSPI*nm*>_Start must be called before this API function is called.

[Syntax]

```
MD_STATUS   R_<Config_MSPInm>_Send (const uint8_t * tx_buf);
MD_STATUS   R_<Config_MSPInm>_Send (const uint16_t * tx_buf);
MD_STATUS   R_<Config_MSPInm>_Send (const uint32_t * tx_buf);
```

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

[Argument(s)]

I/O	Argument(s)	Description
I	const uint8_t * *tx_buf*	Pointer to a buffer storing the transmission data (1 byte-level)
I	const uint16_t * *tx_buf*	Pointer to a buffer storing the transmission data (2 byte-level)
I	const uint32_t * *tx_buf*	Pointer to a buffer storing the transmission data (4 byte-level)

[Return value]

Macro	Description
MD_OK	Normal completion

R_<Config_MSPI*nm*>_Receive

Starts MSPI*nm* data reception.

Remark1. This API function performs MSPIn 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_MSPI*nm*>_Start is then called.

[Syntax]

```
MD_STATUS   R_<Config_MSPInm>_Receive(uint8_t * rx_buf);
MD_STATUS   R_<Config_MSPInm>_Receive(uint16_t * rx_buf);
MD_STATUS   R_<Config_MSPInm>_Receive(uint32_t * rx_buf);
```

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

[Argument(s)]

I/O	Argument(s)	Description
O	uint8_t * *rx_buf*	Pointer to a buffer storing the transmission data(1 byte-level)
O	uint16_t * *rx_ buf*	Pointer to a buffer storing the transmission data(2 byte-level)
O	Uint32_t * *rx_ buf*	Pointer to a buffer storing the transmission data(4 byte-level)

[Return value]

Macro	Description
MD_OK	Normal completion

R_<Config_MSPI*nm*>_Software_Trigger

Sets channel enable and start trigger.

[Syntax]

```
void    R_<Config_MSPInm>_Software_Trigger(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_MSPI*nm*>_Create_UserInit

Performs user-defined initialization relating to the MSPIn functions.

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

[Syntax]

```
void R_<Config_MSPInm>_Create_UserInit(void);
```

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_MSPI*nm*>_channel*nm*_interrupt_send

MSPI*n* channel *m* send interrupt.

[Syntax]

```
void    r_<Config_MSPInm>_channelnm_interrupt_send(void);
```

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_MSPI*nm*>_channel*nm*_interrupt_receive

MSPI*n* channel *m* receive interrupt.

[Syntax]

```
void    r_<Config_MSPInm>_channelnm_interrupt_receive(void);
```

Remark *n* is the unit number, *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 functions is called as the r_mspi*n*_interrupt_send callback routine.

[Syntax]

```
void    r_<Config_MSPInm>_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 MSPI*nm* reception interrupt.

Remark This API functions is called as the r_mspi*n*_interrupt_receive callback routine.

[Syntax]

```
void r_<Config_MSPInm>_Callback_Interrupt_Receive(void);
```

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_MSPI*nm*>_Callback_Interrupt_Error

Performs processing in response to the MSPI*nm* error interrupt.

Remark This API functions is called as the r_mspi*n*_interrupt_error callback routine.

[Syntax]

```
void r_<Config_MSPInm>_Callback_Interrupt_Error(void);
```

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_MSPI*nm*>_Callback_Interrupt_Frameend

Performs processing in response to the MSPI*nm* frameend interrupt.

Remark This API functions is called as the r_mspi*n*_interrupt_frameend callback routine.

[Syntax]

```
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 r_<Config_MSPI*nm*>_Callback_Interrupt_Send corresponding to the MSPIn communication interrupt.

[Syntax]

```
void    r_<Config_MSPInm>_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 MSPIn error interrupt.

[Syntax]

```
void    r_<Config_MSPInm>_callback_receiveend(void);
```

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

[Argument(s)]

None.

[Return value]

None.

r_<Config_MSPI*nm*>_callback_error

Performs processing in response to the MSPI*nm* error interrupt.

Remark This API function is called as the callback routine of interrupt process r_<Config_MSPI*nm*>_Callback_Interrupt_Error corresponding to the MSPIn error interrupt.

[Syntax]

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

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

[Argument(s)]

I/O	Argument(s)	Description
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

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_MSPI1_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]

API Function Name	Description
R_<Config_MSPI*n*>_Create	Performs initialization necessary to control the MSPI Slave functions
R_<Config_MSPI*n*>_Start	Enables MSPI*n*
R_<Config_MSPI*n*>_Stop	Disables MSPI*n*
R_<Config_MSPI*n*>_Send	Starts MSPI*n* data transmission
R_<Config_MSPI*n*>_Receive	Starts MSPI*n* data reception
R_<Config_MSPI*n*>_Software_Trigger	Set channel enable and start trigger
R_<Config_MSPI*n*>_Create_UserInit	Performs user-defined initialization relating to the MSPI Slave functions
r_<Config_MSPI*n*>_channeln0_interrupt_send	Performs processing in response to the MSPI*n* channel 0 send interrupt
r_<Config_MSPI*n*>_channeln0_interrupt_receive	Performs processing in response to the MSPI*n* channel 0 receive interrupt
r_<Config_MSPI*n*>_interrupt_send	Performs processing in response to the MSPI Slave communication interrupt
r_<Config_MSPI*n*>_interrupt_receive	Performs processing in response to the MSPI*n* reception interrupt
r_<Config_MSPI*n*>_interrupt_error	Performs processing in response to the MSPI*n* error interrupt
r_<Config_MSPI*n*>_interrupt_frameend	Performs processing in response to the MSPI*n* frame end interrupt
r_<Config_MSPI*n*>_callback_sendend	Performs processing in response to the MSPI*n* communication interrupt
r_<Config_MSPI*n*>_callback_receiveend	Performs processing in response to the MSPI*n* reception interrupt
r_<Config_MSPI*n*>_callback_error	Performs processing in response to the MSPI*n* error interrupt

R_<Config_MSPI*n*>_Create

Performs initialization necessary to control the MSPI Slave functions.

Remark This API function is called by R_Systeminit function.

[Syntax]

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

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_MSPI*n*>_Start

Enables MSPI*n*.

[Syntax]

```
void    R_<Config_MSPIn>_Start(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_MSPI*n*>_Stop

Disables MSPI*n*.

[Syntax]

```
void    R_<Config_MSPIn>_Stop(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_MSPI*n*>_Send

Starts MSPI*n* data transmission.

Remark1 This API function repeats the MSPIn transmission from the buffer specified in argument tx_buf the number of times specified in frame count.

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

[Syntax]

```
MD_STATUS   R_<Config_MSPIn>_Send (const uint8_t * tx_buf);
MD_STATUS   R_<Config_MSPIn>_Send (const uint16_t * tx_buf);
MD_STATUS   R_<Config_MSPIn>_Send (const uint32_t * tx_buf);
```

Remark *n* is the unit number.

[Argument(s)]

I/O	Argument(s)	Description
I	const uint8_t * *tx_buf*	Pointer to a buffer storing the transmission data (1 byte-level)
I	const uint16_t * *tx_buf*	Pointer to a buffer storing the transmission data (2 byte-level)
I	const uint32_t * *tx_buf*	Pointer to a buffer storing the transmission data (4 byte-level)

[Return value]

Macro	Description
MD_OK	Normal completion

R_<Config_MSPI*n*>_Receive

Starts MSPI*n* data reception.

Remark1 This API function performs MSPIn 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_MSPIn>_Receive (uint8_t * rx_buf);
MD_STATUS   R_<Config_MSPIn>_Receive (uint16_t * rx_buf);
MD_STATUS   R_<Config_MSPIn>_Receive (uint32_t * rx_buf);
```

Remark *n* is the unit number.

[Argument(s)]

I/O	Argument(s)	Description
O	uint8_t * *rx_buf*	Pointer to a buffer storing the transmission data (1 byte-level)
O	uint16_t * *rx_ buf*	Pointer to a buffer storing the transmission data (2 byte-level)
O	Uint32_t * *rx_ buf*	Pointer to a buffer storing the transmission data (4 byte-level)

[Return value]

Macro	Description
MD_OK	Normal completion

R_<Config_MSPI*n*>_Software_Trigger

Sets channel enable and start trigger.

[Syntax]

```
void    R_<Config_MSPIn>_Software_Trigger(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_MSPI*n*>_Create_UserInit

Performs user-defined initialization relating to the MSPI*n* functions.

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

[Syntax]

```
void R_<Config_MSPIn>_Create_UserInit(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_MSPI*n*>_channel*n0*_interrupt_send

Performs processing in response to the MSPI*n* channel 0 send interrupt.

[Syntax]

```
void    r_<Config_MSPIn>_channeln0_interrupt_send(void);
```

Remark *n* is 0, 1.

[Argument(s)]

None.

[Return value]

None.

r_<Config_MSPI*n*>_channel*n0*_interrupt_receive

Performs processing in response to the MSPI*n* channel 0 receive interrupt.

[Syntax]

```
void    r_<Config_MSPIn>_channeln0_interrupt_receive(void);
```

Remark *n* is 0, 1.

[Argument(s)]

None.

[Return value]

None.

r_<Config_MSPI*n*>_interrupt_send

Performs processing in response to the MSPI*n* communication interrupt.

[Syntax]

```
void    r_<Config_MSPIn>_interrupt_send(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_MSPI*n*>_interrupt_receive

Performs processing in response to the MSPI*n* reception interrupt.

[Syntax]

```
void    r_<Config_MSPIn>_interrupt_receive(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_MSPI*n*>_interrupt_error

Performs processing in response to the MSPI*n* error interrupt.

[Syntax]

```
void    r_<Config_MSPIn>_interrupt_error(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_MSPI*n*>_interrupt_frameend

Performs processing in response to the MSPI*n* frame end interrupt.

[Syntax]

```
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*n* communication interrupt.

Remark This API function is called as the callback routine of interrupt process r_<Config_MSPI*n*>_interrupt_send corresponding to the MSPI*n* communication interrupt.

[Syntax]

```
void    r_<Config_MSPIn>_callback_sendend(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_MSPI*n*>_callback_receiveend

Performs processing in response to the MSPI*n* reception interrupt.

Remark This API function is called as the callback routine of interrupt process r_<Config_MSPI*n*>_interrupt_receive corresponding to the MSPI*n* error interrupt.

[Syntax]

```
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 MSPIn error interrupt.

[Syntax]

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

Remark *n* is the unit number.

[Argument(s)]

I/O	Argument(s)	Description
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]

API Function Name	Description
R_<Config_RTCA*n*>_Create	Performs initialization necessary to control the real-time clock
R_<Config_RTCA*n*>_Start	Starts the count of the real-time clock (year, month, weekday, day, hour, minute, second)
R_<Config_RTCA*n*>_Stop	Ends the count of the real-time clock (year, month, weekday, day, hour, minute, second)
R_<Config_RTCA*n*>_Set_HourSystem	Sets the clock type (12-hour or 24-hour clock) of the real-time clock
R_<Config_RTCA*n*>_Get_CounterBufferValue	Reads counter buffer register
R_<Config_RTCA*n*>_Get_CounterDirectValue	Reads the counter register directly
R_<Config_RTCA*n*>_Set_CounterValue	Sets the counter value (year, month, weekday, day, hour, minute, second) of the real-time clock
R_<Config_RTCA*n*>_Get_SubCounterValue	Reads the buffer register of the sub-counter
R_<Config_RTCA*n*>_Set_ErrorCorrectionValue	Sets the error correction value
R_<Config_RTCA*n*>_Set_SubCounterCompareValue	Sets the comparison value of the sub-counter
R_<Config_RTCA*n*>_Get_AlarmValue	Reads the alarm conditions (weekday, hour, minute)
R_<Config_RTCA*n*>_Set_AlarmValue	Sets the alarm conditions (weekday, hour, minute)
R_<Config_RTCA*n*>_Set_AlarmOn	Starts the alarm interrupt function
R_<Config_RTCA*n*>_Set_AlarmOff	Ends the alarm interrupt function
R_<Config_RTCA*n*>_Set_ConstPeriodInterruptOn	Sets the cycle of the periodic interrupts, then starts the periodic interrupt function
R_<Config_RTCA*n*>_Set_ConstPeriodInterruptOff	Ends the periodic interrupt function
R_<Config_RTCA*n*>_Set_1secondInterruptOn	Starts the 1 second interrupt function
R_<Config_RTCA*n*>_Set_1secondInterruptOff	Ends the 1 second interrupt function
R_<Config_RTCA*n*>_Set_RTCA1HZOn	Enables output of the correction clock (1 Hz) to the RTC1HZ pin
R_<Config_RTCA*n*>_Set_RTCA1HZOff	Disables output of the correction clock (1 Hz) to the RTC1HZ pin
R_<Config_RTCA*n*>_Create_UserInit	Performs user-defined initialization relating to the real-time clock
r_<Config_RTCA*n*>_interrupt_alarm	Performs processing in response to the alarm interrupt
r_<Config_RTCA*n*>_interrupt_periodic	Performs processing in response to the periodic interrupt
r_<Config_RTCA*n*>_interrupt_1second	Performs processing in response to the 1 second interrupt

R_<Config_RTCA*n*>_Create

Performs initialization necessary to control the real-time clock.

Remark This API function is called by R_Systeminit function.

[Syntax]

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

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_RTCA*n*>_Start

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

[Syntax]

```
void    R_<Config_RTCAn>_Start(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_RTCA*n*>_Stop

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

[Syntax]

```
void    R_<Config_RTCAn>_Stop(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_RTCA*n*>_Set_HourSystem

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

[Syntax]

MD_STATUS R_<Config_RTCA*n*>_Set_HourSystem(rtc_hour_system_t *hour_system*);

Remark *n* is the unit number.

[Argument(s)]

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

[Return value]

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

R_<Config_RTCA*n*>_Get_CounterBufferValue

Reads counter buffer register.

[Syntax]

```
MD_STATUS    R_<Config_RTCAn>_Get_CounterBufferValue(rtc_counter_value_t * const counter_read_val);
```

Remark *n* is the unit number.

[Argument(s)]

I/O	Argument(s)	Description
O	rtc_counter_value_t * const *counter_read_val*	Pointer to the structure storing the read count value

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

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

[Return value]

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

R_<Config_RTCA*n*>_Get_CounterDirectValue

Reads the counter register directly.

[Syntax]

MD_STATUS R_<Config_RTCA*n*>_Get_CounterDirectValue(rtc_counter_value_t * const *counter_read_val*);

Remark *n* is the unit number.

[Argument(s)]

I/O	Argument(s)	Description
O	rtc_counter_value_t * const *counter_read_val*	Pointer to the structure storing the read count value

Remark For structure rtc_counter_value_t, see R_<Config_RTCAn>_Get_CounterBufferValue.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR	Mismatch between first and second read values

R_<Config_RTCA*n*>_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)]

I/O	Argument(s)	Description
I	rtc_counter_value_t *counter_write_val*	Counter value (year, month, weekday, day, hour, minute, second)

Remark For structure rtc_counter_value_t, see R_<Config_RTCAn>_Get_CounterBufferValue.

[Return value]

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

R_<Config_RTCA*n*>_Get_SubCounterValue

Reads the buffer register of the sub-counter.

[Syntax]

MD_STATUS R_<Config_RTCA*n*>_Get_SubCounterValue(uint32_t * const *subcounter_read_val*);

Remark *n* is the unit number.

[Argument(s)]

I/O	Argument(s)	Description
O	uint32_t * const *subcounter_read_val*	Pointer to the structure storing the read sub-count value

[Return value]

Macro	Description
MD_OK	Normal completion
MD_BUSY	Transfer incomplete

R_<Config_RTCA*n*>_Set_ErrorCorrectionValue

Set the error correction value.

[Syntax]

MD_STATUS R_<Config_RTCA*n*>_Set_ErrorCorrectionValue(uint8_t * const *errorcorrection_write_val*);

Remark *n* is the unit number.

[Argument(s)]

I/O	Argument(s)	Description
I	uint8_t * const *errorcorrection_write_val*	Pointer to the structure storing the error correction value

[Return value]

Macro	Description
MD_OK	Normal completion
MD_BUSY1	Operation stopped status
MD_BUSY2	Write incomplete

R_<Config_RTCA*n*>_Set_SubCounterCompareValue

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)]

I/O	Argument(s)	Description
I	uint32_t * const *subcompare_write_val*	Pointer to the structure storing the comparison value

[Return value]

Macro	Description
MD_OK	Normal completion
MD_BUSY1	Operation stopped status
MD_BUSY2	Write incomplete

R_<Config_RTCA*n*>_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)]

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

Remark Below is shown the structure rtc_alarm_value_t (alarm conditions).

```
typedef struct <
    uint8_t alarmwm;        /* minute */
    uint8_t alarmwh;        /* hour */
    uint8_t alarmww;        /* weekday */
> rtc_alarm_value_t;
```

[Return value]

None.

R_<Config_RTCA*n*>_Set_AlarmValue

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

[Syntax]

```
void  R_<Config_RTCAn>_Set_AlarmValue(rtc_alarm_value_t alarm_val);
```

Remark *n* is the unit number.

[Argument(s)]

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

Remark For structure rtc_alarm_value_t, see R_<Config_RTCAn>_Get_AlarmValue.

[Return value]

None.

R_<Config_RTCA*n*>_Set_AlarmOn

Starts the alarm interrupt function.

[Syntax]

```
void    R_<Config_RTCAn>_Set_AlarmOn(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_RTCA*n*>_Set_AlarmOff

Ends the alarm interrupt function.

[Syntax]

```
void    R_<Config_RTCAn>_Set_AlarmOff(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_RTCA*n*>_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)]

I/O	Argument(s)	Description
I	rtc_int_period_t *period*	Interrupt INTRTC cycle QUARTERSEC : 0.25 seconds HALFSEC : 0.5 seconds ONESEC : 1 second ONEMIN : 1 minute ONEHOUR : 1 hour ONEDAY : 1 day ONEMONTH : 1 month

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

R_<Config_RTCA*n*>_Set_ConstPeriodInterruptOff

Ends the periodic interrupt function.

[Syntax]

```
void    R_<Config_RTCAn>_Set_ConstPeriodInterruptOff(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_RTCA*n*>_Set_1secondInterruptOn

Starts the 1 second interrupt function.

[Syntax]

```
void    R_<Config_RTCAn>_Set_1secondInterruptOn(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_RTCA*n*>_Set_1secondInterruptOff

Ends the 1 second interrupt function.

[Syntax]

```
void    R_<Config_RTCAn>_Set_1secondInterruptOff(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_RTCA*n*>_Set_RTCA1HZOn

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

[Syntax]

```
void    R_<Config_RTCAn>_Set_RTCA1HZOn(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_RTCA*n*>_Set_RTCA1HZOff

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_RTCA*n*>_Create_UserInit

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

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

[Syntax]

```
void R_<Config_RTCAn>_Create_UserInit(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_RTCA*n*>_interrupt_alarm

Performs processing in response to the alarm interrupt.

[Syntax]

```
void    r_<Config_RTCAn>_interrupt_alarm(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_RTCA*n*>_interrupt_periodic

Performs processing in response to the periodic interrupt.

[Syntax]

```
void    r_<Config_RTCAn>_interrupt_periodic(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_RTCA*n*>_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]

API Function Name	Description
R_<Config_ECM>_Create	Performs initialization necessary to control the error control module
R_<Config_ECM>_Start	Starts the error control module
R_<Config_ECM>_Stop	Stops the error control module
R_<Config_ECM>_Set_Master_Error	Sets master error
R_<Config_ECM>_Set_Checker_Error	Sets checker error
R_<Config_ECM>_Clear_Master_Error	Clear master error
R_<Config_ECM>_Clear_Checker_Error	Clear checker error
R_<Config_ECM>_Delay_Timer_Start	Starts the delay timer
R_<Config_ECM>_Delay_Timer_Stop	Stops the delay timer
R_<Config_ECM>_Pseud _*XXXX*_Start	Starts ECM pseudo debug operation for *XXXX* function
R_<Config_ECM>_Pseudo_*XXXX*_Stop	Stops ECM pseudo debug operation for *XXXX* function
R_<Config_ECM>_Create_UserInit	Performs user-defined initialization relating to the error control module functions
r_<Config_ECM>_ecmmi_interrupt_pe*k*	Performs processing in response to the ECM maskable interrupt for PE *k*
r_<Config_ECM>_mi_interrupt_pe*k*	Performs processing in response to the ECM maskable interrupt for PE *k*
r_<Config_ECM>_dclsmi_interrupt_pe*k*	Performs processing in response to the DCLS error interrupt for PE *k*

R_<Config_ECM>_Create

Performs initialization necessary to control the error control module.

Remark This API function is called by R_Systeminit function.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ECM>_Start

Starts the error control module.

[Syntax]

```
void    R_<Config_ECM>_Start(void);
```

[Argument(s)]

None.

[Return value]

None.

R_<Config_ECM>_Stop

Stops the error control module.

[Syntax]

```
void    R_<Config_ECM>_Stop(void);
```

[Argument(s)]

None.

[Return value]

None.

R_<Config_ECM>_Set_Master_Error

Sets the master error.

[Syntax]

```
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]

```
void    R_<Config_ECM>_Clear_Master_Error(void);
```

[Argument(s)]

None.

[Return value]

None.

R_<Config_ECM>_Clear_Checker_Error

Clears the checker error.

[Syntax]

```
void    R_<Config_ECM>_Clear_Checker_Error(void);
```

[Argument(s)]

None.

[Return value]

None.

R_<Config_ECM>_Delay_Timer_Start

Starts the delay timer.

[Syntax]

```
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]

```
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]

```
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

	XXXX name		XXXX name
1	Delay_Timer_Overflow	2	ECM_Compare_Error
3	Activation_Production_Test_Mode	4	Activation_Normal_Operation_Mode
5	Deactivation_Normal_Operation_Mode	6	Serial_Programming_Mode
7	Activation_User_Boot_Mode	8	Deactivation_User_Boot_Mode
9	Mode_Check_Error	10	Flash_Access_Error
11	FACI_Reset_Transfer_Error	12	BIST_Parameter_Transfer_Error
13	DTS_Compare_Error	14	BUS_Bridge_Compare_Error_SDMAC
15	BUS_Bridge_Compare_Error	16	Element_Bus_Routing_Error
17	I_Bus_Routing_Error	18	P_Bus_Routing_Error
19	CRAM_Bus_Routing_Error	20	System_Bus_Routing_Error
21	Global_Flash_Bus_Routing_Error	22	Local_Flash_Bus_Routing_Error
23	Clock_Monitor_Error_MOSC	24	Clock_Monitor_Error_HSintOSC
25	Clock_Monitor_Error_LSintOSC	26	Clock_Monitor_Error_CLK_LSB
27	Clock_Monitor_Error_CLK_UHSB	28	Clock_Monitor_Error_CLK_HBUS
29	OSTM*n*_Interrupt_Error	30	ADCJ_Parity_Error
31	Temperature_Sensor_Error	32	Code_Flash_Address_Parity_Error
33	Code_Flash_ECC_Uncorrectable_Error	34	Code_Flash_ECC_Correctable_Error
35	Code_Flash_ECC_Overflow_Error	36	Data_Flash_ECC_Uncorrectable_Error
37	Data_Flash_ECC_Correctable_Error	38	Data_Flash_ECC_Overflow_Error
39	Local_RAM_ECC_Uncorrectable_Error	40	Local_RAM_ECC_Correctable_Error
41	Local_RAM_ECC_Overflow_Error	42	Cluster_RAM_ECC_Uncorrectable_Error
43	Cluster_RAM_ECC_Correctable_Error	44	Cluster_RAM_ECC_Overflow_Error
45	DTSRAM_ECC_Uncorrectable_Error	46	DTSRAM_ECC_Correctable_Error
47	DTSRAM_ECC_Overflow_Error	48	SDMACnRAM_ECC_Uncorrectable_Error
49	SDMACnRAM_ECC_Correctable_Error	50	FlexRayRAM_ECC_Uncorrectable_Error
51	FlexRayRAM_ECC_Correctable_Error	52	RS_CANFDRAM_ECC_Uncorrectable_Error
53	RS_CANFDRAM_ECC_Correctable_Error	54	MSPI_RAM_ECC_Uncorrectable_Error
55	MSPI_RAM _ECC_Correctable_Error	56	GTM_RAM_ECC_Uncorrectable_Error
57	GTM_RAM _ECC_Correctable_Error	58	Fast_Ethernet_RAM_ECC_Uncorrectable_Error
59	Fast_Ethernet_RAM_ECC_Correctable_Error	60	Gigabit_Ethernet_RAM_ECC_Uncorrectable_Error
61	Gigabit_Ethernet_RAM_ECC_Correctable_Error	62	MMCA_RAM_ECC_Uncorrectable_Error
63	MMCA_RAM_ECC_Correctable_Error	64	Peripheral_ECC_Overflow_Error
65	Data_Transfer_Path_EDC_Error	66	Data_Transfer_Path_ECC_Uncorrectable_Error
67	Data_Transfer_Path_ECC_Correctable_Error	68	CRAM_Guard_Error

69	I_Bus_Guard_Error	70	P_Bus_Guard_Error
71	H_Bus_Guard_Error	72	DTS_sDMAC_Transfer_Error
73	H_Bus_Master_Transfer_Error	74	External_Error_Input_*n*_Error
75	Software_Alarm_*n*_Error	76	DCLS_Compare_Error_PE*k*
77	Unintended_Debug_Enable_Detection_PE*k*	78	Watchdog_Timer_Error_PE*k*
79	Clock_Monitor_Error_PE*k*	80	PE_Guard_Error_PE*k*
81	Data_Access_Error_PE*k*	82	PE_Guard_Error_PE*k*_Access
83	Local_RAM_PE*k*_ECC_Uncorrectable_Error	84	Local_RAM_PE*k*_ECC_Correctable_Error
85	Local_RAM_PE*k*_ECC_Overflow_Error	86	Instruction_Cache_RAM_PE*k*_ECC_Uncorrectable_Error

[Argument(s)]

None.

[Return value]

None.

R_<Config_ECM>_Pseudo_*XXXX*_Stop

Stops ECM pseudo debug operation for XXXX function.

[Syntax]

```
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]

```
void    R_<Config_ECM>_Create_UserInit(void);
```

[Argument(s)]

None.

[Return value]

None.

r_<Config_ECM>_ecmmi_interrupt_pe*k*

Performs processing in response to the ECM maskable interrupt for PE *k.*

[Syntax]

```
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_pe*k*

Performs processing in response to the ECM maskable interrupt for PE *k*.

[Syntax]

```
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_pe*k*

Performs processing in response to the DCLS error interrupt for PE *k*.

[Syntax]

```
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]

API Function Name	Description
R_<Config_GTM_Clock>_Create	Performs initialization necessary to control the GTM clock
R_ GTM_CCMn_Create	Performs initialization necessary to control the cluster configuration module
R_<Config_GTM_Clock>_Create_UserInit	Performs user-defined initialization relating to the GTM clock

R_<Config_GTM_Clock>_Create

Performs initialization necessary to control the GTM clock.

Remark This API function is called by R_Systeminit function.

[Syntax]

```
void    R_<Config_GTM_Clock>_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 R_<Config_GTM_Clock>_Create callback routine.

[Syntax]

```
void R_GTM_CCMn_Create(void);
```

Remark *n* is the CCM channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_GTM_Clock>_Create_UserInit

Performs user-defined initialization relating to the GTM clock.

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

[Syntax]

```
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]

API Function Name	Description
R_<Config_TBU>_Create	Performs initialization necessary to control the time base unit functions
R_<Config_TBU>_Start	Starts the count for time base unit
R_<Config_TBU>_Stop	Ends the count for time base unit
R_<Config_TBU>_Create_UserInit	Performs user-defined initialization relating to the time base unit functions

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]

```
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]

```
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 functions is called as the R_<Config_TBU>_Create callback routine.

[Syntax]

```
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]

API Function Name	Description
R_<Config_TIM*n*>_Create	Performs initialization necessary to control the TIM bit compression mode
R_<Config_TIM*n*>_Start	Starts compression processing for unit *n*
R_<Config_TIM*n*>_Stop	Stops compression processing for unit *n*
R_<Config_TIM*n*>_SetDetectedEdge	Sets detected edge number
R_<Config_TIM*n*>_Software_Reset	Stops channel operation and reset channel registers
R_<Config_TIM*n*>_NEWVAL_TriggerOn	Trigger NEWVAL bit in TIMn_CH0_IRQ_NOTIFY register by software
R_<Config_TIM*n*>_ECNTOFL_TriggerOn	Trigger ECNTOFL bit in TIMn_CH0_IRQ_NOTIFY register by software
R_<Config_TIM*n*>_CNTOFL_TriggerOn	Trigger CNTOFL bit in TIMn_CH0_IRQ_NOTIFY register by software
R_<Config_TIM*n*>_GPROFL_TriggerOn	Trigger GPROFL bit in TIMn_CH0_IRQ_NOTIFY register by software
R_<Config_TIM*n*>_TODETOFL_TriggerOn	Trigger TODETOFL bit in TIMn_CH0_IRQ_NOTIFY register by software
R_<Config_TIM*n*>_Channel*m*_GLITCHDETOFL_TriggerOn	Trigger GLITCHDET bit in TIMn_CHm_IRQ_NOTIFY register by software
R_<Config_TIM*n*>_Create_UserInit	Performs user-defined initialization relating to the TIM bit compression mode
r_<Config_TIM*n*>_Callback_GTM_Error	Processing is performed according to the occurrence of a count error interrupt
r_<Config_TIM*n*>_share_interrupt	Performs processing in response to the TIM*n* interrupt
r_<Config_TIM*n*>_ch*m*_share_interrupt	Performs processing in response to the TIM*n* channel*m* interrupt

R_<Config_TIM*n*>_Create

Performs initialization necessary to control the TIM bit compression mode.

Remark This API function is called by R_Systeminit function.

[Syntax]

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

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_TIM*n*>_Start

Starts compression processing for unit *n*.

[Syntax]

```
void    R_<Config_TIMn>_Start(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_TIM*n*>_Stop

Stops compression processing for unit *n*.

[Syntax]

```
void    R_<Config_TIMn>_Stop(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_TIM*n*>_SetDetectedEdge

Sets detected edge number.

[Syntax]

```
void    R_<Config_ TIMn>_ SetDetectedEdge (uint16_t number);
```

Remark *n* is the unit number.

[Argument(s)]

I/O	Argument(s)	Description
I	uint16_t number	Detected edge number

[Return value]

None.

R_<Config_TIM*n*>_Software_Reset

Stops channel operation and reset channel registers.

[Syntax]

```
void    R_<Config_TIMn>_Software_Reset(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_TIM*n*>_NEWVAL_TriggerOn

Trigger NEWVAL bit in TIMn_CH0_IRQ_NOTIFY register by software.

[Syntax]

```
void    R_<Config_TIMn>_NEWVAL_TriggerOn(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_TIM*n*>_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_TIM*n*>_CNTOFL_TriggerOn

Trigger CNTOFL bit in TIMn_CH0_IRQ_NOTIFY register by software.

[Syntax]

```
void    R_<Config_TIMn>_CNTOFL_TriggerOn(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_TIM*n*>_GPROFL_TriggerOn

Trigger GPROFL bit in TIMn_CH0_IRQ_NOTIFY register by software.

[Syntax]

```
void    R_<Config_TIMn>_GPROFL_TriggerOn(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_TIM*n*>_TODETOFL_TriggerOn

Trigger TODETOFL bit in TIM*n*_CH0_IRQ_NOTIFY register by software.

[Syntax]

```
void    R_<Config_TIMn>_TODETOFL_TriggerOn(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_TIM*n*>_Channel*m*_GLITCHDETOFL_TriggerOn

Trigger GLITCHDETOFL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software

[Syntax]

```
void    R_<Config_TIMn>_Channelm_GLITCHDETOFL_TriggerOn(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_TIM*n*>_Create_UserInit

Performs user-defined initialization relating to the TIM bit compression mode.

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

[Syntax]

```
void R_<Config_TIMn>_Create_UserInit(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_TIM*n*>_Callback_GTM_Error

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]

```
void    r_<Config_TIMn>_Callback_GTM_Error(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_TIM*n*>_share_interrupt

Performs processing in response to the TIM*n* interrupt.

[Syntax]

```
void    r_<Config_TIMn>_share_interrupt(void);
```

Remark *n* is the unit number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_TIM*n*>_ch*m*_share_interrupt

Performs processing in response to the TIM*n* channel*m* interrupt.

[Syntax]

```
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 3.43 API Functions: [TIM Gated Periodic Sampling Mode]

API Function Name	Description
R_<Config_TIM*n*_*m*>_Create	Performs initialization necessary to control the TIM gated periodic sampling mode.
R_<Config_TIM*n*_*m*>_Start	Starts periodic sampling for channel *m* in unit *n*
R_<Config_TIM*n*_*m*>_Stop	Stops periodic sampling for channel *m* in unit *n*
R_<Config_TIM*n*_*m*>_UpdateElapsedClock	Updates elapsed clock number.
R_<Config_TIM*n*_*m*>_Software_Reset	Stops channel operation and reset channel registers
R_<Config_TIM*n*_*m*>_NEWVAL_TriggerOn	Trigger NEWVAL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_ECNTOFL_TriggerOn	Trigger ECNTOFL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_CNTOFL_TriggerOn	Trigger CNTOFL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_GPROFL_TriggerOn	Trigger GPROFL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_TODETOFL_TriggerOn	Trigger TODETOFL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_GLITCHDETOFL_TriggerOn	Trigger GLITCHDET bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_Create_UserInit	Performs user-defined initialization relating to the TIM gated periodic sampling mode.
r_<Config_TIM*n*_*m*>_Callback_GTM_Error	Processing is performed according to the occurrence of a count error interrupt.
r_<Config_TIM*n*_*m*>_share_interrupt	Performs processing in response to the TIMn channel*m* interrupt.

R_<Config_TIM*n*_*m*>_Create

Performs initialization necessary to control the TIM gated periodic sampling 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_TIM*n*_*m*>_Start

Starts periodic sampling for channel *m* in unit *n*.

[Syntax]

```
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_TIM*n*_*m*>_Stop

Stops periodic sampling 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_TIM*n_m*>_UpdateElapsedClock

Updates elapsed clock number.

[Syntax]

```
void    R_<Config_ TIMn_m>_ UpdateElapsedClock (uint32_t number);
```

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

[Argument(s)]

I/O	Argument(s)	Description
I	Uint32_t number	Elapsed clock number

[Return value]

None.

R_<Config_TIM*n*_*m*>_Software_Reset

Stops channel operation and reset channel registers.

[Syntax]

```
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 TIM*n*_CH*m*_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_TIM*n*_*m*>_ECNTOFL_TriggerOn

Trigger ECNTOFL bit in TIM*n*_CH*m*_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_TIM*n*_*m*>_CNTOFL_TriggerOn

Trigger CNTOFL bit in TIM*n*_CH*m*_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_<Config_TIM*n*_*m*>_GPROFL_TriggerOn

Trigger GPROFL bit in TIM*n*_CH*m*_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_TIM*n*_*m*>_TODETOFL_TriggerOn

Trigger TODETOFL bit in TIM*n*_CH*m*_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_TIM*n*_*m*>_GLITCHDETOFL_TriggerOn

Trigger GLITCHDETOFL bit in TIM*n*_CH*m*_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_TIM*n_m*>_Create_UserInit

Performs user-defined initialization relating to the TIM gated periodic sampling mode.

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

[Syntax]

```
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_TIM*n_m*>_ Callback_GTM_Error

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]

```
void    r_<Config_TIMn_m>_ Callback_GTM_Error(void);
```

[Argument(s)]

None.

[Return value]

None.

r_<Config_TIM*n*_*m*>_share_interrupt

Performs processing in response to the TIM*n* channel*m* 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 3.44 API Functions: [TIM Input Prescaler Mode]

API Function Name	Description
R_<Config_TIM*n*_*m*>_Create	Performs initialization necessary to control the TIM input prescaler mode
R_<Config_TIM*n*_*m*>_Start	Starts the count for channel *m* in unit *n*
R_<Config_TIM*n*_*m*>_Stop	Stops the count for channel *m* in unit *n*
R_<Config_TIM*n*_*m*>_SetDetectedEdge	Sets detected edge number
R_<Config_TIM*n*_*m*>_Software_Reset	Stops channel operation and reset channel registers
R_<Config_TIM*n*_*m*>_NEWVAL_TriggerOn	Trigger NEWVAL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_ECNTOFL_TriggerOn	Trigger ECNTOFL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_CNTOFL_TriggerOn	Trigger CNTOFL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_GPROFL_TriggerOn	Trigger GPROFL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_TODETOFL_TriggerOn	Trigger TODETOFL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_GLITCHDETOFL_TriggerOn	Trigger GLITCHDET bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_Create_UserInit	Performs user-defined initialization relating to the TIM input prescaler mode
r_<Config_TIM*n*_*m*>_Callback_GTM_Error	Processing is performed according to the occurrence of a count error interrupt
r_<Config_TIM*n*_*m*>_share_interrupt	Performs processing in response to the TIM*n* channel*m* interrupt

R_<Config_TIM*n*_*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_TIM*n*_*m*>_Start

Starts the count for channel *m* in unit *n*.

[Syntax]

```
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_TIM*n*_*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_TIM*n_m*>_SetDetectedEdge

Sets detected edge number.

[Syntax]

```
void    R_<Config_ TIMn_m>_ SetDetectedEdge (uint32_t number);
```

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

[Argument(s)]

I/O	Argument(s)	Description
I	Uint32_t number	Detected edge number

[Return value]

None.

R_<Config_TIM*n*_*m*>_Software_Reset

Stops channel operation and reset channel registers.

[Syntax]

```
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 TIM*n*_CH*m*_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_TIM*n*_*m*>_ECNTOFL_TriggerOn

Trigger ECNTOFL bit in TIM*n*_CH*m*_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_TIM*n*_*m*>_CNTOFL_TriggerOn

Trigger CNTOFL bit in TIM*n*_CH*m*_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_<Config_TIM*n*_*m*>_GPROFL_TriggerOn

Trigger GPROFL bit in TIM*n*_CH*m*_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_TIM*n*_*m*>_TODETOFL_TriggerOn

Trigger TODETOFL bit in TIM*n*_CH*m*_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_TIM*n*_*m*>_GLITCHDETOFL_TriggerOn

Trigger GLITCHDETOFL bit in TIM*n*_CH*m*_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_TIM*n_m*>_Create_UserInit

Performs user-defined initialization relating to the TIM input prescaler mode.

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

[Syntax]

```
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_TIM*n*_*m*>_Callback_GTM_Error

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]

```
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_TIM*n*_*m*>_share_interrupt

Performs processing in response to the TIM*n* channel*m* 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.

3.2.45 TIM Input Event Mode

Below is a list of API functions output by the Code Generator for TIM input event mode use.

Table 3.45 API Functions: [TIM Input Prescaler Mode]

API Function Name	Description
R_<Config_TIM*n*_*m*>_Create	Performs initialization necessary to control the TIM input event mode
R_<Config_TIM*n*_*m*>_Start	Starts the count for channel *m* in unit *n*
R_<Config_TIM*n*_*m*>_Stop	Stops the count for channel *m* in unit *n*
R_<Config_TIM*n*_*m*>_Software_Reset	Stops channel operation and reset channel registers
R_<Config_TIM*n*_*m*>_NEWVAL_TriggerOn	Trigger NEWVAL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_ECNTOFL_TriggerOn	Trigger ECNTOFL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_CNTOFL_TriggerOn	Trigger CNTOFL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_GPROFL_TriggerOn	Trigger GPROFL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_TODETOFL_TriggerOn	Trigger TODETOFL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_GLITCHDETOFL_TriggerOn	Trigger GLITCHDET bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_Create_UserInit	Performs user-defined initialization relating to the TIM input event mode
r_<Config_TIM*n*_*m*>_Callback_GTM_Error	Processing is performed according to the occurrence of a count error interrupt
r_<Config_TIM*n*_*m*>_share_interrupt	Performs processing in response to the TIM*n* channel*m* interrupt

R_<Config_TIM*n*_*m*>_Create

Performs initialization necessary to control the TIM input event 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_TIM*n*_*m*>_Start

Starts the count for channel *m* in unit In.

[Syntax]

```
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_TIM*n*_*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_TIM*n*_*m*>_Software_Reset

Stops channel operation and reset channel registers.

[Syntax]

```
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 TIM*n*_CH*m*_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_TIM*n*_*m*>_ECNTOFL_TriggerOn

Trigger ECNTOFL bit in TIM*n*_CH*m*_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_TIM*n*_*m*>_CNTOFL_TriggerOn

Trigger CNTOFL bit in TIM*n*_CH*m*_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_<Config_TIM*n*_*m*>_GPROFL_TriggerOn

Trigger GPROFL bit in TIM*n*_CH*m*_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_TIM*n*_*m*>_TODETOFL_TriggerOn

Trigger TODETOFL bit in TIM*n*_CH*m*_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_TIM*n*_*m*>_GLITCHDETOFL_TriggerOn

Trigger GLITCHDETOFL bit in TIM*n*_CH*m*_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_TIM*n_m*>_Create_UserInit

Performs user-defined initialization relating to the TIM input event mode.

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

[Syntax]

```
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_TIM*n*_*m*>_Callback_GTM_Error

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]

```
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_TIM*n*_*m*>_share_interrupt

Performs processing in response to the TIM*n* channel*m* 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.

3.2.46 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]

API Function Name	Description
R_<Config_TIM*n*_*m*>_Create	Performs initialization necessary to control the TIM pulse integration mode
R_<Config_TIM*n*_*m*>_Start	Starts the count for channel *m* in unit *n*
R_<Config_TIM*n*_*m*>_Stop	Stops the count for channel *m* in unit *n*
R_<Config_TIM*n*_*m*>_Software_Reset	Stops channel operation and reset channel registers
R_<Config_TIM*n*_*m*>_NEWVAL_TriggerOn	Trigger NEWVAL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_ECNTOFL_TriggerOn	Trigger ECNTOFL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_CNTOFL_TriggerOn	Trigger CNTOFL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_GPROFL_TriggerOn	Trigger GPROFL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_TODETOFL_TriggerOn	Trigger TODETOFL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_GLITCHDETOFL_TriggerOn	Trigger GLITCHDET bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_Create_UserInit	Performs user-defined initialization relating to the TIM pulse integration mode
r_<Config_TIM*n*_*m*>_Callback_GTM_Error	Processing is performed according to the occurrence of a count error interrupt
r_<Config_TIM*n*_*m*>_share_interrupt	Performs processing in response to the TIM*n* channel*m* interrupt

R_<Config_TIM*n*_*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_TIM*n*_*m*>_Start

Starts the count for channel *m* in unit *n*.

[Syntax]

```
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_TIM*n*_*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_TIM*n*_*m*>_Software_Reset

Stops channel operation and reset channel registers.

[Syntax]

```
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 TIM*n*_CH*m*_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_TIM*n*_*m*>_ECNTOFL_TriggerOn

Trigger ECNTOFL bit in TIM*n*_CH*m*_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_TIM*n*_*m*>_CNTOFL_TriggerOn

Trigger CNTOFL bit in TIM*n*_CH*m*_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_<Config_TIM*n*_*m*>_GPROFL_TriggerOn

Trigger GPROFL bit in TIM*n*_CH*m*_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_TIM*n*_*m*>_TODETOFL_TriggerOn

Trigger TODETOFL bit in TIM*n*_CH*m*_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_TIM*n*_*m*>_GLITCHDETOFL_TriggerOn

Trigger GLITCHDETOFL bit in TIM*n*_CH*m*_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_TIM*n_m*>_Create_UserInit

Performs user-defined initialization relating to the TIM pulse integration mode.

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

[Syntax]

```
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_TIM*n*_*m*>_Callback_GTM_Error

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]

```
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_TIM*n*_*m*>_share_interrupt

Performs processing in response to the TIM*n* channel*m* 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.

3.2.47 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]

API Function Name	Description
R_<Config_TIM*n*_*m*>_Create	Performs initialization necessary to control the TIM pwm measurement mode
R_<Config_TIM*n*_*m*>_Start	Starts the count for channel *m* in unit *n*
R_<Config_TIM*n*_*m*>_Stop	Stops the count for channel *m* in unit *n*
R_<Config_TIM*n*_*m*>_Software_Reset	Stops channel operation and reset channel registers
R_<Config_TIM*n*_*m*>_NEWVAL_TriggerOn	Trigger NEWVAL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_ECNTOFL_TriggerOn	Trigger ECNTOFL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_CNTOFL_TriggerOn	Trigger CNTOFL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_GPROFL_TriggerOn	Trigger GPROFL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_TODETOFL_TriggerOn	Trigger TODETOFL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_GLITCHDETOFL_TriggerOn	Trigger GLITCHDET bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_Create_UserInit	Performs user-defined initialization relating to the TIM pwm measurement mode
r_<Config_TIM*n*_*m*>_Callback_GTM_Error	Processing is performed according to the occurrence of a count error interrupt
r_<Config_TIM*n*_*m*>_share_interrupt	Performs processing in response to the TIM*n* channel*m* interrupt

R_<Config_TIM*n*_*m*>_Create

Performs initialization necessary to control the TIM PWM measurement 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_TIM*n*_*m*>_Start

Starts the count for channel *m* in unit *n*.

[Syntax]

```
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_TIM*n*_*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_TIM*n*_*m*>_Software_Reset

Stops channel operation and reset channel registers.

[Syntax]

```
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 TIM*n*_CH*m*_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_TIM*n*_*m*>_ECNTOFL_TriggerOn

Trigger ECNTOFL bit in TIM*n*_CH*m*_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_TIM*n*_*m*>_CNTOFL_TriggerOn

Trigger CNTOFL bit in TIM*n*_CH*m*_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_<Config_TIM*n*_*m*>_GPROFL_TriggerOn

Trigger GPROFL bit in TIM*n*_CH*m*_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_TIM*n*_*m*>_TODETOFL_TriggerOn

Trigger TODETOFL bit in TIM*n*_CH*m*_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_TIM*n*_*m*>_GLITCHDETOFL_TriggerOn

Trigger GLITCHDETOFL bit in TIM*n*_CH*m*_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_TIM*n_m*>_Create_UserInit

Performs user-defined initialization relating to the TIM PWM measurement mode.

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

[Syntax]

```
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_TIM*n*_*m*>_Callback_GTM_Error

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]

```
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_TIM*n*_*m*>_share_interrupt

Performs processing in response to the TIM*n* channel*m* 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.

3.2.48 TIM Serial Shift Mode

Below is a list of API functions output by the Code Generator for TIM serial shift mode use.

Table 3.48 API Functions: [TIM serial shift mode]

API Function Name	Description
R_<Config_TIM*n*_*m*>_Create	Performs initialization necessary to control the TIM serial shift mode.
R_<Config_TIM*n*_*m*>_Start	Starts serial shift for channel *m* in unit *n*
R_<Config_TIM*n*_*m*>_Stop	Stops serial shift for channel *m* in unit *n*
R_<Config_TIM*n*_*m*>_UpdateElapsedClock	Updates elapsed clock number.
R_<Config_TIM*n*_*m*>_UpdateShiftClock	Updates shift clock number
R_<Config_TIM*n*_*m*>_UpdateCaptrueSource	Updates capture source
R_<Config_TIM*n*_*m*>_UpdateTssmOut	Updates tssm out signal
R_<Config_TIM*n*_*m*>_SetGPR0AsShadowRegister	Sets TIM*n*_CH*m*_GPR0 as a shadow register for TIM*n*_CH*m*_CNT
R_<Config_TIM*n*_*m*>_Software_Reset	Stops channel operation and reset channel registers
R_<Config_TIM*n*_*m*>_NEWVAL_TriggerOn	Trigger NEWVAL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_ECNTOFL_TriggerOn	Trigger ECNTOFL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_CNTOFL_TriggerOn	Trigger CNTOFL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_GPROFL_TriggerOn	Trigger GPROFL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_TODETOFL_TriggerOn	Trigger TODETOFL bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_GLITCHDETOFL_TriggerOn	Trigger GLITCHDET bit in TIM*n*_CH*m*_IRQ_NOTIFY register by software
R_<Config_TIM*n*_*m*>_Create_UserInit	Performs user-defined initialization relating to the TIM serial shift mode.
r_<Config_TIM*n*_*m*>_Callback_GTM_Error	Processing is performed according to the occurrence of a count error interrupt.
r_<Config_TIM*n*_*m*>_share_interrupt	Performs processing in response to the TIM*n* channel*m* interrupt.

R_<Config_TIM*n*_*m*>_Create

Performs initialization necessary to control the TIM serial shift 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_TIM*n*_*m*>_Start

Starts the count for channel *m* in unit In.

[Syntax]

```
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_TIM*n*_*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_TIM*n*_*m*>_UpdateElapsedClock

Updates elapsed clock number.

[Syntax]

```
void    R_<Config_TIMn_m>_UpdateElapsedClock(uint8_t number);
```

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

[Argument(s)]

I/O	Argument(s)	Description
I	Uint8_t number	Elapsed clock number

[Return value]

None.

R_<Config_TIM*n*_*m*>_UpdateShiftClock

Updates shift clock number.

[Syntax]

```
void    R_<Config_TIMn_m>_ UpdateShiftClock(uint8_t number);
```

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

[Argument(s)]

I/O	Argument(s)	Description
I	Uint8_t number	Shift clock number

[Return value]

None.

R_<Config_TIM*n_m*>_UpdateCaptrueSource

Updates capture source.

[Syntax]

```
void    R_<Config_TIMn_m>_UpdateCaptrueSource(uint8_t number);
```

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

[Argument(s)]

I/O	Argument(s)	Description
I	Uint8_t number	Capture source

[Return value]

None.

R_<Config_TIM*n*_*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)]

I/O	Argument(s)	Description
I	Uint8_t number;	Tssm out signal

[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]

```
void    R_<Config_TIMn_m>_SetGPR0AsShadowRegister(char number);
```

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

[Argument(s)]

I/O	Argument(s)	Description
I	char number	Sets whether to use GPR0 as the shadow register of the CNT register

[Return value]

None.

R_<Config_TIM*n*_*m*>_Software_Reset

Stops channel operation and reset channel registers.

[Syntax]

```
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 TIM*n*_CH*m*_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_TIM*n*_*m*>_ECNTOFL_TriggerOn

Trigger ECNTOFL bit in TIM*n*_CH*m*_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_TIM*n*_*m*>_CNTOFL_TriggerOn

Trigger CNTOFL bit in TIM*n*_CH*m*_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_<Config_TIM*n*_*m*>_GPROFL_TriggerOn

Trigger GPROFL bit in TIM*n*_CH*m*_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_TIM*n*_*m*>_TODETOFL_TriggerOn

Trigger TODETOFL bit in TIM*n*_CH*m*_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_TIM*n*_*m*>_GLITCHDETOFL_TriggerOn

Trigger GLITCHDETOFL bit in TIM*n*_CH*m*_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_TIM*n_m*>_Create_UserInit

Performs user-defined initialization relating to the TIM serial shift mode.

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

[Syntax]

```
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_TIM*n*_*m*>_Callback_GTM_Error

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]

```
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_TIM*n*_*m*>_share_interrupt

Performs processing in response to the TIM*n* channel*m* 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 3.49 API Functions: [ATOM Signal Output Mode Compare]

API Function Name	Description
R_<Config_ATOM*n*_*m*>_Create	Performs initialization necessary to control the ATOM signal output mode compare
R_<Config_ATOM*n*_*m*>_Start	Starts ATOM*nm* signal output mode compare
R_<Config_ATOM*n*_*m*>_Stop	Stop ATOM*nm* signal output mode compare
R_<Config_ATOM*n*_*m*>_Output_Start	Enable ATOM*nm* output function of signal output mode compare
R_<Config_ATOM*n*_*m*>_Output_Stop	Disable ATOM*nm* output function of signal output mode compare
R_<Config_ATOM*n*_*m*>_Channel_Reset	Software reset of ATOM*nm* channel
R_<Config_ATOM*n*_*m*>_LateUpdate_Request	CPU Write request for ATOM*nm* late compare register update
R_<Config_ATOM*n*_*m*>_Create_UserInit	Performs user-defined initialization relating to the ATOM signal output mode compare functions
R_<Config_ATOM*n*_*m*>_Callback_Shared_IRQ*k*	Performs processing in response to the ATOM*nm* shared interrupt

R_<Config_ATOM*n*_*m*>_Create

Performs initialization necessary to control the ATOM signal output mode compare.

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_ATOM*n*_*m*>_Start

Starts ATOMnm signal output mode compare.

[Syntax]

```
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_ATOM*n*_*m*>_Stop

Stop ATOM*nm* signal output mode compare.

[Syntax]

```
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_<Config_ATOM*n*_*m*>_Output_Start

Enable ATOM*nm* output function of signal output mode compare.

[Syntax]

```
void    R_<Config_ATOMn_m>_Output_Start(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ATOM*n*_*m*>_Output_Stop

Disable ATOM*nm* output function of signal output mode compare.

[Syntax]

```
void    R_<Config_ATOMn_m>_Output_Stop(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ATOM*n*_*m*>_Channel_Reset

Software reset of ATOM*nm* channel.

[Syntax]

```
void    R_<Config_ATOMn_m>_Channel_Reset(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ATOM*n*_*m*>_LateUpdate_Request

CPU Write request for ATOM*nm* late compare register update.

[Syntax]

```
void    R_<Config_ATOMn_m>_LateUpdate_Request(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ATOM*n*_*m*>_Create_UserInit

Performs user-defined initialization relating to the ATOM signal output mode compare functions.

Remark This API functions is called as the R_<Config_ATOM*n*_*m*>_Create callback routine.

[Syntax]

```
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_IRQ*k*

Performs processing in response to the ATOM*nm* shared interrupt.

Remark This API function is called by r_atom*n*_irq*k*_interrupt function.

[Syntax]

```
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]

API Function Name	Description
R_<Config_ATOM*n*_*m*>_Create	Performs initialization necessary to control the ATOM signal output mode immediate
R_<Config_ATOM*n*_*m*>_Start	Starts ATOM*nm* signal output mode immediate
R_<Config_ATOM*n*_*m*>_Stop	Stop ATOM*nm* signal output mode immediate
R_<Config_ATOM*n*_*m*>_Output_Start	Enable ATOM*nm* output function of signal output mode immediate
R_<Config_ATOM*n*_*m*>_Output_Stop	Disable ATOM*nm* output function of signal output mode immediate
R_<Config_ATOM*n*_*m*>_Channel_Reset	Software reset of ATOM*nm* channel
R_<Config_ATOM*n*_*m*>_Create_UserInit	Performs user-defined initialization relating to the ATOM signal output mode immediate functions

R_<Config_ATOM*n*_*m*>_Create

Performs initialization necessary to control the ATOM signal output mode immediate.

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_ATOM*n*_*m*>_Start

Starts ATOM*nm* signal output mode immediate.

[Syntax]

```
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_ATOM*n*_*m*>_Stop

Stop ATOM*nm* signal output mode immediate.

[Syntax]

```
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_<Config_ATOM*n*_*m*>_Output_Start

Enable ATOM*nm* output function of signal output mode immediate.

[Syntax]

```
void    R_<Config_ATOMn_m>_Output_Start(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ATOM*n*_*m*>_Output_Stop

Disable ATOM*nm* output function of signal output mode immediate.

[Syntax]

```
void    R_<Config_ATOMn_m>_Output_Stop(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ATOM*n*_*m*>_Channel_Reset

Software reset of ATOM*nm* channel.

[Syntax]

```
void    R_<Config_ATOMn_m>_Channel_Reset(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ATOM*n*_*m*>_Create_UserInit

Performs user-defined initialization relating to the ATOM signal output mode immediate functions.

Remark This API functions is called as the R_<Config_ATOM*n*_*m*>_Create callback routine.

[Syntax]

```
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.

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]

API Function Name	Description
R_<Config_ATOM*n_m*>_Create	Performs initialization necessary to control the ATOM signal output mode PWM
R_<Config_ATOM*n_m*>_Start	Starts ATOM*nm* signal output mode PWM
R_<Config_ATOM*n_m*>_Stop	Stop ATOM*nm* signal output mode PWM
R_<Config_ATOM*n_m*>_Output_Start	Enable ATOM*nm* output function of signal output mode PWM
R_<Config_ATOMn_m>_Output_Stop	Disable ATOM*nm* output function of signal output mode PWM
R_<Config_ATOMn_m>_Channel_Reset	Software reset of ATOM*nm* channel
R_<Config_ATOMn_m>_Create_UserInit	Performs user-defined initialization relating to the ATOM signal output mode PWM functions
R_<Config_ATOM*n_m*>_Callback_Shared_IRQ*k*	Performs processing in response to the ATOM*nm* shared interrupt

R_<Config_ATOM*n*_*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_ATOM*n*_*m*>_Start

Starts ATOM*nm* signal output mode PWM.

[Syntax]

```
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_ATOM*n*_*m*>_Stop

Stop ATOM*nm* signal output mode PWM.

[Syntax]

```
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_<Config_ATOM*n*_*m*>_Output_Start

Enable ATOM*nm* output function of signal output mode PWM.

[Syntax]

```
void    R_<Config_ATOMn_m>_Output_Start(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ATOM*n*_*m*>_Output_Stop

Disable ATOM*nm* output function of signal output mode PWM.

[Syntax]

```
void    R_<Config_ATOMn_m>_Output_Stop(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ATOM*n*_*m*>_Channel_Reset

Software reset of ATOM*nm* channel.

[Syntax]

```
void    R_<Config_ATOMn_m>_Channel_Reset(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ATOM*n*_*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_ATOM*n*_*m*>_Create callback routine.

[Syntax]

```
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_IRQ*k*

Performs processing in response to the ATOM*nm* shared interrupt.

Remark This API function is called by r_atom*n*_irq*k*_interrupt function.

[Syntax]

```
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]

API Function Name	Description
R_<Config_ATOM*n*_*m*>_Create	Performs initialization necessary to control the ATOM signal output mode serial
R_<Config_ATOM*n*_*m*>_Start	Starts ATOM*nm* signal output mode serial
R_<Config_ATOM*n*_*m*>_Stop	Stop ATOM*nm* signal output mode serial
R_<Config_ATOM*n*_*m*>_Output_Start	Enable ATOM*nm* output function of signal output mode serial
R_<Config_ATOMn_m>_Output_Stop	Disable ATOM*nm* output function of signal output mode serial
R_<Config_ATOMn_m>_Channel_Reset	Software reset of ATOM*nm* channel
R_<Config_ATOM*n*_*m*>_Create_UserInit	Performs user-defined initialization relating to the ATOM signal output mode serial functions
R_<Config_ATOM*n*_*m*>_Callback_Shared_IRQ*k*	Performs processing in response to the ATOM*nm* shared interrupt

R_<Config_ATOM*n*_*m*>_Create

Performs initialization necessary to control the ATOM signal output mode serial.

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_ATOM*n*_*m*>_Start

Starts ATOM*nm* signal output mode serial.

[Syntax]

```
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_ATOM*n*_*m*>_Stop

Stop ATOM*nm* signal output mode serial.

[Syntax]

```
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_<Config_ATOM*n*_*m*>_Output_Start

Enable ATOM*nm* output function of signal output mode serial.

[Syntax]

```
void    R_<Config_ATOMn_m>_Output_Start(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ATOM*n*_*m*>_Output_Stop

Disable ATOM*nm* output function of signal output mode serial.

[Syntax]

```
void    R_<Config_ATOMn_m>_Output_Stop(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ATOM*n*_*m*>_Channel_Reset

Software reset of ATOM*nm* channel.

[Syntax]

```
void    R_<Config_ATOMn_m>_Channel_Reset(void);
```

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ATOM*n*_*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_ATOM*n*_*m*>_Create callback routine.

[Syntax]

```
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_IRQ*k*

Performs processing in response to the ATOM*nm* shared interrupt.

Remark This API function is called by r_atom*n*_irq*k*_interrupt function.

[Syntax]

```
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.

3.2.53 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]

API Function Name	Description
R_<Config_CDTM*n_m*>_Create	Performs initialization necessary to control the dead time module
R_<Config_CDTM*n_m*>_Create_UserInit	Performs user-defined initialization relating to the dead time module

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]

```
void R_<Config_CDTMn_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 functions is called as the R_<Config_CDTM*n*_*m*>_Create callback routine.

[Syntax]

```
void R_ <Config_CDTMn_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]

API Function Name	Description
R_<Config_DTS*n*>_Create	Performs initialization necessary to control the DTS functions which includes chain header channel*n* and chained channel
R_<Config_DTS*n*>_Start	Enables the DTS operation which inldues chain header channel*n* and chained channel
R_<Config_DTS*n*>_Stop	Disables the DTS operation which includes chain header channel*n* and chained channel
R_<Config_DTS*n*>_Set_SoftwareTrigger	Set DTS software trigger for chain header channel*n*
r_<Config_DTS*n*>_Callback_Dts*g*_Transfer_End	Performs processing in response to the DTS channel *g* transfer end interrupt
r_<Config_DTS*n*>_Callback_Dts*g*_Transfer_Count_Match	Performs processing in response to the DTS channel *g* transfer count match interrupt
r_<Config_DTS*n*>_Callback_PE*k*_Transfer_Error	Performs processing in response to the DTS transfer error interrupt for PE*k*
r_<Config_DTS*n*>_Callback_Transfer_Error	Performs processing in response to the DTS transfer error interrupt
R_<Config_DTS*n*>_Create_UserInit	Performs user-defined initialization relating to DTS controller functions

R_<Config_DTS*n*>_Create

Performs initialization necessary to control the DTS*n* 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_DTS*n*>_Start

Enables the DTS operation.

[Syntax]

```
void    R_<Config_DTSn>_Start(void);
```

Remark *n* is the chain channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_DTS*n*>_Stop

Disables the DTS operation.

[Syntax]

```
void    R_<Config_DTSn>_Stop(void);
```

Remark *n* is the chain header channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_DTS*n*>_Set_SoftwareTrigger

Generates the DTS channel *n* transfer request.

[Syntax]

```
void    R_<Config_DTSn>_Set_SoftwareTrigger(void);
```

Remark *n* is the chain header channel number.

[Argument(s)]

None.

[Return value]

None.

R_<Config_DTS*n*>_Create_UserInit

Performs user-defined initialization relating to the DTS functions.

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

[Syntax]

```
void R_<Config_DTSn>_Create_UserInit(void);
```

Remark *n* is the chain header channel number.

[Argument(s)]

None.

[Return value]

None.

r_<Config_DTS*n*>_Callback_Dts*g*_Transfer_End

Performs processing in response to the DTS transfer end interrupt.

Remark This API functions is called by r_dts*g*_transfer_end_interrupt routine.

[Syntax]

```
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_DTS*n*>_Callback_Dts*g*_Transfer_Count_Match

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

Remark This API functions is called by r_dts*g*_transfer_count_match_interrupt routine.

[Syntax]

```
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_DTS*n*>_Callback_PE*k*_Transfer_Error

Performs processing in response to the DTS transfer error interrupt.

Remark This API functions is called by r_dts_transfer_error_interrupt_pek routine.

[Syntax]

void r_<Config_DTS*n*>_Callback_PE*k*_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_DTS*n*>_Callback_Transfer_Error

Performs processing in response to the DTS transfer error interrupt.

Remark This API functions is called by r_dmac_error_interrupt_pe*k* 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]

API Function Name	Description
R_<Config_ABFG>_Create	Performs initialization necessary to control the ADC Boundary Flag Generator functions
R_<Config_ABFG>_Start	Starts ABFG operation
R_<Config_ABFG>_Stop	Stops ABFG operation
R_<Config_ABFG>_Create_UserInit	Performs user-defined initialization relating to the ADC Boundary Flag Generator functions
r_<Config_ABFG>_boundary_flag_pulse_*w*_interrupt	Performs processing in response to the interrupt request signal from AIR

R_<Config_ABFG>_Create

Performs initialization necessary to control the ADC Boundary Flag Generator functions.

Remark This API function is called by R_Systeminit function.

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

R_<Config_ABFG>_Start

Starts ABFG operation.

[Syntax]

```
void    R_<Config_ABFG>_Start(void);
```

[Argument(s)]

None.

[Return value]

None.

R_<Config_ABFG>_Stop

Stops ABFG operation.

[Syntax]

```
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 functions is called as the R_<Config_ABFG>_Create callback routine.

[Syntax]

```
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

Rev.	Section	Description
1.00	All	First Edition issued
1.10	3. API FUNCTIONS	Table 3.1.1 API Functions: [General]: Add new APIs
		Add chapter 3.2.34 to 3.2.54
1.20	All	Add "<>" for all configuration name in API name
	2. OUTPUT FILES	Table 2.1 Output File List: Remove all private functions
		Table 2.1 Output File List: Change the "start" and "stop" API name for Clock Divide function.
		Table 2.1 Output File List: Delete duplicated API name for DMA controller function.
		Table 2.1 Output File List: Add "r_<Config_ATOMn_m>_callback_shared_interrupt" for ATOM Signal Output Mode PWM and ATOM Signal Output Mode Serial function
		Table 2.1 Output File List: Change the API name in [Syntax] part.
	3. API FUNCTIONS	3.2.6 Input Interval Timer 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
		3.2.7 Input Pulse Interval 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
		3.2.8 Interval Timer 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
		3.2.9 Triangle 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
		3.2.10 Triangle PWM Output With Dead Time 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

Rev.	Section	Description
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

Rev.	Section	Description
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

Rev.	Section	Description
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_MSPInm>_enable_interrupt r_<Config_MSPInm>_disable_interrupt
		3.2.37 MSPI Slave Remove the following private functions from this chapter r_<Config_MSPIn>_enable_interrupt r_<Config_MSPIn>_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

Rev.	Section	Description
1.20	3. API FUNCTIONS	3.2.47 TIM PWM Measurement Mode Remove the following private functions from this chapter r_<Config_TIMn_m>_enable_interrupt r_<Config_TIMn_m>_disable_interrupt
		3.2.48 TIM Serial Shift Mode Remove the following private functions from this chapter r_<Config_TIMn_m>_enable_interrupt r_<Config_TIMn_m>_disable_interrupt
		3.2.49 ATOM Signal Output Mode Compare Remove the following private functions from this chapter r_<Config_ATOMn_m>_enable_interrupt r_<Config_ATOMn_m>_disable_interrupt
		3.2.51 ATOM Signal Output Mode PWM Remove the following private functions from this chapter r_<Config_ATOMn_m>_enable_interrupt r_<Config_ATOMn_m>_disable_interrupt
		3.2.52 ATOM Signal Output Mode Serial Remove the following private functions from this chapter r_<Config_ATOMn_m>_enable_interrupt r_<Config_ATOMn_m>_disable_interrupt
		3.2.36 MSPI Master R_<Config_MSPInm>_Send: Deleter the redundant “const” in parameter in API [Syntax] part. R_<Config_MSPInm>_Receive: Change the description of API
		3.2.39 Error Control Module R_<Config_ECM>_Pseudo_XXXX_Start and R_<Config_ECM>_Pseudo_XXXX_Stop: Change the API name in [Syntax] part.
		Table 3.39.1 ECM pseudo error name: Change the table name. Change “ADCJ0_Parity_Error” to “ADCJn_Parity_Error” Add “Remark” in [Syntax] part.
1.30	3. API FUNCTIONS	3.2.3 CSI Slave: R_<Config_CSI*Xn*>_Send: add “const” for parameter “tx_buf” R_<Config_CSIXn>_Receive: add “const” for parameter “rx_buf”
		3.2.4 CSI Master: Add function “R_<Config_CSIHn>_Send_Receive” R_<Config_CSIXn>_Send: add “const” for parameter “tx_buf” R_<Config_CSI*Xn*>_Receive: add “const” for parameter “rx_buf”
		3.2.4 CSI Master: Add function “R_<Config_CSIHn>_Send_Receive” R_<Config_CSIXn>_Send: add “const” for parameter “tx_buf” R_<Config_CSI*Xn*>_Receive: add “const” for parameter “rx_buf”
		3.2.15 UART Interface R_<Config_UART*n*>_Send, R_<Config_UART*n*>_Receive: Add Remark3.

Rev.	Section	Description
1.30	3. API FUNCTIONS	3.2.34 SCI3 Clock Synchronous Mode R_<Config_SCI3n>_Send: adjust “const” position of parameter “tx_buf” R_<Config_SCI3n>_Receive: add “const” for parameter “rx_buf”
		3.2.35 SCI3 Asynchronous Mode R_<Config_SCI3n>_Send: adjust “const” position of parameter “tx_buf” R_<Config_SCI3n>_Receive: add “const” for parameter “rx_buf” R_<Config_SCI3n>_Multiprocessor_Send: adjust “const” position of parameter “tx_buf” R_<Config_SCI3n>_Multiprocessor_Receive: add “const” for parameter “rx_buf”
1.40	2. OUTPUT FILES	Table 2.1 Output File List: Add the following files. r_cg_ad_air.h r_cg_abfg.h r_cg_ad_common.c r_cg_ad_common.h
	3. API FUNCTIONS	3.2.1 General: The following functions are added: R_ADC_SyncStart, R_ADC_TimerSyncStart, R_DMAC_Create, R_DMA_Suspend, R_DMA_Resume, R_PDMAn_Suspend, R_PDMAn_Resume, R_DTS_Create, R_DMA_Start_Transfer_Error_Interrupt, R_DMA_Stop_Transfer_Error_Interrupt, r_dmac_error_interrupt_pe1, R_MSPI_Master_Create Following function Remark description changed: R_GTM_Start_Interrupt_Error R_GTM_Stop_Interrupt_Error R_ATOM*n*_Start_Interrupt_IRQ*k* R_ATOM*n*_Stop_Interrupt_IRQ*k*

Rev.	Section	Description
1.40	3. API FUNCTIONS	3.2.2 A/D Converter The following functions are added: R_<Config_ADCXn>_ScanGroupx_OperationOff, R_<Config_ADCXn>_TH_Sampling_Stop R_<Config_ADCXn>_TH_Stop R_<Config_ADCXn>_VoltageDivider_Start R_<Config_ADCXn>_VoltageDivider_Stop R_<Config_ADCXn>_StrongPullDown_Start R_<Config_ADCXn>_StrongPullDown_Stop R_<Config_ADCXn>_SGDiag_Start R_<Config_ADCXn>_SGDiag_OperationOn R_<Config_ADCXn>_SGDiag_OperationOff R_<Config_ADCXn>_SGDiag_GetResult R_<Config_ADCXn>_SGDiag_GetSubtractionResult R_<Config_ADCXn>_ASF_Start R_<Config_ADCXn>_ASF_Stop R_<Config_ADCXn>_ASF_GetResult R_<Config_ADCXn>_ASF_GetRealTimeResult r_<Config_ADCXn>_sg_diag_end_interrupt r_<Config_ADCXn>_mpx_request_interrupt r_<Config_ADCXn>_asf_channelk_end_interrupt Following functions are removed: R_<Config_ADCXn>_SyncStart R_<Config_ADCXn>_SyncTimerStart Following function name is updated: Old: R_<Config_ADCXn>_ScanGroupx_GetFloatingPointDataResult, New: R_<Config_ADCXn>_ScanGroupx_GetFloatingPointResult, Following function parameter is updated: R_<Config_ADCXn>_ScanGroupx_GetFloatingPointResult, R_<Config_ADC*Xn*>_ScanGroup*x*_GetResult Following function description is updated: R_<Config_ADC*Xn*>_ScanGroup*x*_OperationOn R_<Config_ADC*Xn*>_ScanGroup*x*_OperationOff
		3.2.3 CSI Slave R_<Config_CSIXn>_Send_Receive is added to support send and receive function
		3.2.4 CSI Master R_<Config_CSIXn>_Send_Receive is added to support send and receive function

Rev.	Section	Description
1.40	3. API FUNCTIONS	3.2.5 Interrupt Following functions are added: R_<Config_INTC>_IRQn_Start R_<Config_INTC>_IRQn_Stop r_<Config_INTC>_intpn_interrupt_pek r_<Config_INTC>_irqn_interrupt
		3.2.14 Stand-by Controller Following functions are added: R_<Config_STBC>_Prepare_Stop_Mode_Set_CPUCLK R_<Config_STBC>_Prepare_Deep_Stop_Mode_Set_Clock_Mask R_<Config_STBC>_Prepare_Deep_Stop_Mode_Set_Clock_Source
		3.2.16 Watchdog Timer Following functions are added: R_<Config_WDTAn>_Create R_<Config_WDTAn>_Restart R_<Config_WDTAn>_Create_UserInit R_<Config_WDTAn>_Inerrupt
		3.2.18 Data CRC Following functions are added: R_<Config_DCRAn>_InitializeCRCData R_<Config_KCRCn>_InitializeCRCData
		3.2.20 DMA Controller Following functions are added: r_<Config_DMACnm>_Callback_DMACnm_Transfer_Error R_<Config_SDMACnm>_Reset r_<Config_SDMACnm>_Callback_PEx_Address_Error Following function name is updated: Old: r_<Config_DMACnm>_interrupt New: r_<Config_DMACnm>_dmacnm_interrupt
		3.2.36 MSPI Master Following function name is updated: Old: r_<Config_MSPInm>_callback_send r_<Config_MSPInm>_callback_receive r_<Config_MSPInm>_interrupt_error r_<Config_MSPInm>_callback_frameend New: r_<Config_MSPInm>_Callback_Interrupt_Send r_<Config_MSPInm>_Callback_Interrupt_Receive r_<Config_MSPInm>_Callback_Interrupt_Error r_<Config_MSPInm>_Callback_Interrupt_Frameend

Rev.	Section	Description
1.40	3. API FUNCTIONS	3.2.39 Error Module Control Following functions are added: R_<Config_ECM>_Set_Master_Error R_<Config_ECM>_Set_Checker_Error R_<Config_ECM>_Clear_Master_Error R_<Config_ECM>_Clear_Checker_Error R_<Config_ECM>_Delay_Timer_Start R_<Config_ECM>_Delay_Timer_Stop r_<Config_ECM>_mi_interrupt_pek
		3.2.42 TIM Bit Compression Mode 3.2.43 TIM Gated Periodic Sampling Mode 3.2.44 TIM Input Prescaler Mode 3.2.45 TIM Input Event Mode 3.2.46 TIM Pulse Integration Mode 3.2.47 TIM PWM Measurement Mode 3.2.48 TIM Serial Shift Mode Following function name updated: Old: r_<Config_TIMn>_callback_gtm_error New: r_<Config_TIMn>_Callback_GTM_Error
		3.2.49 ATOM Signal Output Mode Compare 3.2.51 ATOM Signal Output Mode PWM 3.2.52 ATOM Signal Output Mode Serial Following function name is updated: Old: r_<Config_ATOMn_m>_callback_shared_interrupt New: r_<Config_ATOMn_m>_Callback_Shared_IRQk

Rev.	Section	Description
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_dtsg_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_Dtsg_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

Rev.	Section	Description
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
1.60	3. API FUNCTIONS	3.2.49 ATOM Signal Output Mode Compare Add below new API: R_<Config_ATOMn_m>_Output_Start R_<Config_ATOMn_m>_Output_Stop R_<Config_ATOMn_m>_Channel_Reset R_<Config_ATOMn_m>_LateUpdate_Request 3.2.50 ATOM Signal Output Mode Immediate Add below new API: R_<Config_ATOMn_m>_Output_Start R_<Config_ATOMn_m>_Output_Stop R_<Config_ATOMn_m>_Channel_Reset 3.2.51 ATOM Signal Output Mode PWM Add below new API: R_<Config_ATOMn_m>_Output_Start R_<Config_ATOMn_m>_Output_Stop R_<Config_ATOMn_m>_Channel_Reset 3.2.52 ATOM Signal Output Mode Serial Add below new API: R_<Config_ATOMn_m>_Output_Start R_<Config_ATOMn_m>_Output_Stop R_<Config_ATOMn_m>_Channel_Reset
	3. API FUNCTIONS	Fix 3 same typo error of API for below chapters: 3.2.49 ATOM Signal Output Mode Compare 3.2.51 ATOM Signal Output Mode PWM 3.2.52 ATOM Signal Output Mode Serial r_<Config_ATOM*n_m*>_Callback_Shared_IRQ*k is updated to:* R_<Config_ATOM*n_m*>_Callback_Shared_IRQ*k*

Smart Configurator User's Manual: RH850 API Reference

Publication Date: Rev.1.00 Jul.13.18
Rev.1.60 Jan.20.24
Published by: Renesas Electronics Corporation

Smart Configurator

Renesas Electronics Corporation

R20UT4361EJ0160