User's Manual

スマート・コンフィグレータ

ユーザーズマニュアル　RX API リファレンス編

ターゲットデバイス
RX ファミリー

本資料に記載の全ての情報は本資料発行時点のものであり、ルネサス エレクトロニクスは、
予告なしに、本資料に記載した製品または仕様を変更することがあります。
ルネサス エレクトロニクスのホームページなどにより公開される最新情報をご確認ください。

ルネサス エレクトロニクス
www.renesas.com

Rev.1.08　2023.10

このマニュアルの使い方

対象者　　このマニュアルは、スマート・コンフィグレータのドライバコード生成の機能を理解し、それを用いたアプリケーション・システムを開発するユーザを対象としています。

目的　　このマニュアルは、スマート・コンフィグレータのドライバコード生成の持つソフトウエア機能をユーザに理解していただき、これを使用するシステムのハードウエア、ソフトウエア開発の参照用資料として役立つことを目的としています。

構成　　このマニュアルは、大きく分けて次の内容で構成しています。

読み方　　このマニュアルを読むにあたっては、電気、論理回路、マイクロコンピュータに関する一般知識が必要となります。

凡例			
	データ表記の重み	:	左が上位桁、右が下位桁
	アクティブ・ロウの表記	:	$\overline{\text{XXX}}$　（端子、信号名称に上線）
	注	:	本文中につけた注の説明
	注意	:	気をつけて読んでいただきたい内容
	備考	:	本文中の補足説明
	数の表記	:	10 進数...XXXX
			16 進数...0xXXXX

目次

1. 概説

本章では、スマート・コンフィグレータのドライバコード生成機能の概要について説明します。

1.1 概要

本ツールは、GUI ベースで各種情報を設定することにより、デバイスが提供している周辺機能（クロック発生回路、電圧検出回路など）を制御するうえで必要なソースコード（デバイスドライバプログラム：C ソースファイル，ヘッダファイル）を出力することができます。

1.2 特長

以下に、コード生成機能の特長を示します。

- コード生成機能
  GUI ベースでマイコンの周辺機能を設定し、設定情報に応じた初期化プログラムを出力するだけでなく、動作開始／停止、割り込み関数も出力します。

- レポート機能
  設定した情報を各種形式のファイルで出力し、設計資料として利用することができます。

- リネーム機能
  出力するフォルダ名、ファイル名、およびソースコードに含まれている API 関数の関数名については、デフォルトの名前が付与されますが、ユーザ独自の名前に変更することもできます。
  本文書内では、ユーザが変更可能な箇所を<>で囲んで表記しています。

- ユーザコード保護機能
  各 API 関数には、ユーザが独自にコードを追加できるように、ユーザコード記述用のコメントが設けられています。

  ［ユーザコード記述用のコメント］
  /* Start user code. Do not edit comment generated here */
  /* End user code. Do not edit comment generated here */

  このコメント内にコードを記述すると、再度コード生成した場合でもユーザが記述したコードは保護されます。

2. 出力ファイル

本章では、コード生成が出力するファイルについて説明します。

2.1 説明

以下に、コード生成が出力するファイルの一覧を示します。

表 2.1 出力ファイル(1/12)

周辺機能	ファイル名	API 関数名
共通	<workspaceName>.c	main
	dbsct.c	―
	resetprg.c	PowerON_Reset
	sbrk.c	―
	vecttbl.c	―
	vecttbl.h	―
	hwsetup.c	hardware_setup
	hwsetup.h	―
	r_cg_hardware_setup.c	r_undefined_exception R_Systeminit
	r_cg_macrodriver.h	―
	r_cg_userdefine.h	―
	r_smc_entry.h	―
	r_smc_cgc.c	R_CGC_Create
	r_smc_cgc_user.c	R_CGC_Create_UserInit
	r_smc_cgc.h	―
	r_smc_interrup.c	R_Interrupt_Create
	r_smc_interrupt.h	―
	Pin.c	R_Pins_Create
	Pin.h	―

表 2.2 出力ファイル(2/13)

周辺機能	ファイル名	API 関数名
8 ビットタイマ	<Config_TMR0>.c	R_<Config_TMR0>_Create R_<Config_TMR0>_Start R_<Config_TMR0>_Stop
	<Config_TMR0>_user.c	R_<Config_TMR0>_Create_UserInit r_<Config_TMR0>_cmimn_interrupt r_<Config_TMR0>_ovin_interrupt
	<Config_TMR0>.h	―
バス	<Config_BSC>.c	R_<Config_BSC>_Create R_<Config_BSC>_Error_Monitoring_Start R_<Config_BSC>_Error_Monitoring_Stop R_<Config_BSC>_InitializeSDRAM
	<Config_BSC>_user.c	R_<Config_BSC>_Create_UserInit r_<Config_BSC>_buserr_interrupt
	<Config_BSC>.h	―
クロック周波数精度測定回路	<Config_CAC>.c	R_<Config_CAC>_Create R_<Config_CAC>_Start R_<Config_CAC>_Stop
	<Config_CAC>_user.c	R_<Config_CAC>_Create_UserInit r_<Config_CAC>_mendf_interrupt r_<Config_CAC>_mendi_interrupt r_<Config_CAC>_ferrf_interrupt r_<Config_CAC>_ferri_interrupt r_<Config_CAC>_ovff_interrupt r_<Config_CAC>_ovfi_interrupt
	<Config_CAC>.h	―
コンパレータ	<Config_CMPB0>.c	R_<Config_CMPB0>_Create R_<Config_CMPB0>_Start R_<Config_CMPB0>_Stop
	<Config_CMPB0>user.c	R_<Config_CMPB0>_Create_UserInit r_<Config_CMPB0>_cmpbn_interrupt
	<Config_CMPB0>.h	―
コンペアマッチタイマ	<Config_CMT0>.c	R_<Config_CMT0>_Create R_<Config_CMT0>_Start R_<Config_CMT0>_Stop
	<Config_CMT0>_user.c	R_<Config_CMT0>_Create_UserInit r_<Config_CMT0>_cmin_interrupt r_<Config_CMT0>_cmwin_interrupt r_<Config_CMT0>_icmin_interrupt r_<Config_CMT0>_ocmin_interrupt
	<Config_CMT0>.h	―

表 2.3 出力ファイル(3/13)

周辺機能	ファイル名	API 関数名
相補 PWM モードタイマ	<Config_MTU3_MTU4>.c	R_<Config_MTU3_MTU4>_Create R_<Config_MTU3_MTU4>_Start R_<Config_MTU3_MTU4>_Stop
	<Config_MTU3_MTU4>_user.c	R_<Config_MTU3_MTU4>_Create_UserInit r_<Config_MTU3_MTU4>_tgimn_interrupt r_<Config_MTU3_MTU4>_cj_tgimj_interrupt r_<Config_MTU3_MTU4>_cj_tcivj_interrupt
	<Config_MTU3_MTU4>.h	―
連続スキャンモード S12AD	<Config_S12AD0>.c	R_<Config_S12AD0>_Create R_<Config_S12AD0>_Start R_<Config_S12AD0>_Stop R_<Config_S12AD0>_Get_ValueResult R_<Config_S12AD0>_Set_CompareValue R_<Config_S12AD0>_Set_CompareAValue R_<Config_S12AD0>_Set_CompareBValue
	<Config_S12AD0>_user.c	R_<Config_S12AD0>_Create_UserInit r_<Config_S12AD0>_interrupt r_<Config_S12AD0>_compare_interrupt r_<Config_S12AD0>_compare_interruptA r_<Config_S12AD0>_compare_interruptB
	<Config_S12AD0>.h	―
CRC 演算器	<Config_CRC>.c	R_<Config_CRC>_SetCRC8 R_<Config_CRC>_SetCRC16 R_<Config_CRC>_SetCCITT R_<Config_CRC>_SetCRC32 R_<Config_CRC>_SetCRC32C R_<Config_CRC>_Input_Data R_<Config_CRC>_Get_Result
	<Config_CRC>.h	―
D/A コンバータ	<Config_DA>.c	R_<Config_DA>_Create R_<Config_DA>n_Start R_<Config_DA>n_Stop R_<Config_DA>n_Set_ConversionValue R_<Config_DA>_Sync_Start R_<Config_DA>_Sync_Stop
	<Config_DA>_user.c	R_<Config_DA>_Create_UserInit
	<Config_DA>.h	―

表 2.4 出力ファイル(4/13)

周辺機能	ファイル名	API 関数名
データ演算回路	<Config_DOC.c	R_<Config_DOC>_Create R_<Config_DOC>_SetMode R_<Config_DOC>_WriteData R_<Config_DOC>_GetResult R_<Config_DOC>_ClearFlag
	<Config_DOC_user.c	R_<Config_DOC>_Create_UserInit r_<Config_DOC>_dopcf_interrupt r_<Config_DOC>_dopci_interrupt
	<Config_DOC.h	―
データトランスファコントローラ	<Config_DTC.c	R_<Config_DTC>_Create R_<Config_DTC>_Start R_<Config_DTC>_Stop
	<Config_DTC_user.c	R_<Config_DTC>_Create_UserInit
	<Config_DTC.h	―
デッドタイム補償用カウンタ	<Config_MTU5>.c	R_<Config_MTU5>_Create R_<Config_MTU5>_U5_Start R_<Config_MTU5>_U5_Stop R_<Config_MTU5>_V5_Start R_<Config_MTU5>_V5_Stop R_<Config_MTU5>_W5_Start R_<Config_MTU5>_W5_Stop
	<Config_MTU5>_user.c	R_<Config_MTU5>_Create_UserInit r_<Config_MTU5>_tgimn_interrupt
	<Config_MTU5>.h	―
DMA コントローラ	<Config_DMAC0>.c	R_<Config_DMAC0>_Create R_<Config_DMAC0>_Start R_<Config_DMAC0>_Stop R_<Config_DMAC0>_Set_SoftwareTrigger R_<Config_DMAC0>_Clear_SoftwareTrigger
	<Config_DMAC0>_user.c	R_<Config_DMAC0>_Create_UserInit r_<Config_DMAC0>_dmacni_interrupt r_dmacn_callback_transfer_end r_dmacn_callback_transfer_escape_end
	<Config_DMAC0>.h	―
	r_cg_dmac_user.c	r_dmac_dmac74i_interrupt r_dmacn_callback_transfer_end r_dmacn_callback_transfer_escape_end
	r_cg_dmac.h	―
イベントリンクコントローラ	<Config_ELC>.c	R_<Config_ELC>_Create R_<Config_ELC>_Start R_<Config_ELC>_Stop R_<Config_ELC>_GenerateSoftwareEvent R_<Config_ELC>_Set_PortBuffern R_<Config_ELC>_Get_PortBuffern
	<Config_ELC>_user.c	R_<Config_ELC>_Create_UserInit r_<Config_ELC>_elsrni_interrupt
	<Config_ELC>.h	―

表 2.5 出力ファイル(5/13)

周辺機能	ファイル名	API 関数名
汎用 PWM タイマ	<Config_GPT0>.c	R_<Config_GPT0>_Create R_<Config_GPT0>_Start R_<Config_GPT0>_Stop R_<Config_GPT0>_HardwareStart R_<Config_GPT0>_HardwareStop R_<Config_GPT0>_ETGI_Start R_<Config_GPT0>_ETGI_Stop R_<Config_GPT0>_Software_Clear
	<Config_GPT0>_user.c	R_<Config_GPT0>_Create_UserInit r_<Config_GPT0>_gtcimn_interrupt r_<Config_GPT0>_gtcivn_interrupt r_<Config_GPT0>_gtciun_interrupt r_<Config_GPT0>_gdten_interrupt
	<Config_GPT0>.h	―
	r_cg_gpt_user.c	r_gpt_etgin_interrupt r_gpt_etgip_interrupt
	r_cg_gpt.h	―
グループスキャンモード S12AD	<Config_S12AD0>.c	R_<Config_S12AD0>_Create R_<Config_S12AD0>_Start R_<Config_S12AD0>_Stop R_<Config_S12AD0>_Get_ValueResult R_<Config_S12AD0>_Set_CompareValue R_<Config_S12AD0>_Set_CompareAValue R_<Config_S12AD0>_Set_CompareBValue
	<Config_S12AD0>_user.c	R_<Config_S12AD0>_Create_UserInit r_<Config_S12AD0>_interrupt r_<Config_S12AD0>_compare_interrupt r_<Config_S12AD0>_compare_interruptA r_<Config_S12AD0>_compare_interruptB r_<Config_S12AD0>_groupb_interrupt r_<Config_S12AD0>_groupc_interrupt
	<Config_S12AD0>.h	―

表 2.6 出力ファイル(6/13)

周辺機能	ファイル名	API 関数名
I2C マスタモード	<Config_RIIC0>.c	R_<Config_RIIC0>_Create R_<Config_RIIC0>_Start R_<Config_RIIC0>_Stop R_<Config_RIIC0>_Master_Send R_<Config_RIIC0>_Master_Send_Without_Stop R_<Config_RIIC0>_Master_Receive R_<Config_RIIC0>_IIC_StartCondition R_<Config_RIIC0>_IIC_StopCondition
	<Config_RIIC0>_user.c	R_<Config_RIIC0>_Create_UserInit r_<Config_RIIC0>_error_interrupt r_<Config_RIIC0>_receive_interrupt r_<Config_RIIC0>_transmit_interrupt r_<Config_RIIC0>_transmitend_interrupt r_<Config_RIIC0>_callback_error r_<Config_RIIC0>_callback_transmitend r_<Config_RIIC0>_callback_receiveend
	<Config_RIIC0>.h	―
	<Config_SCI0>.c	R_<Config_SCI0>_Create R_<Config_SCI0>_Start R_<Config_SCI0>_Stop R_<Config_SCI0>_IIC_Master_Send R_<Config_SCI0>_IIC_Master_Receive R_<Config_SCI0>_IIC_StartCondition R_<Config_SCI0>_IIC_StopCondition
	<Config_SCI0>_user.c	R_<Config_SCI0>_Create_UserInit r_<Config_SCI0>_receive_interrupt r_<Config_SCI0>_transmit_interrupt r_<Config_SCI0>_transmitend_interrupt r_<Config_SCI0>_callback_transmitend r_<Config_SCI0>_callback_receiveend
	<Config_SCI0>.h	―
I2C スレーブモード	<Config_RIIC0>.c	R_<Config_RIIC0>_Create R_<Config_RIIC0>_Start R_<Config_RIIC0>_Stop R_<Config_RIIC0>_Slave_Send R_<Config_RIIC0>_Slave_Receive
	<Config_RIIC0>_user.c	R_<Config_RIIC0>_Create_UserInit r_<Config_RIIC0>_error_interrupt r_<Config_RIIC0>_receive_interrupt r_<Config_RIIC0>_transmit_interrupt r_<Config_RIIC0>_transmitend_interrupt r_<Config_RIIC0>_callback_error r_<Config_RIIC0>_callback_transmitend r_<Config_RIIC0>_callback_receiveend
	<Config_RIIC0>.h	―

表 2.7 出力ファイル(7/13)

周辺機能	ファイル名	API 関数名
割り込みコントローラ	<Config_ICU>.c	R_<Config_ICU>_Create R_<Config_ICU>_IRQn_Start R_<Config_ICU>_IRQn_Stop R_<Config_ICU>_Software_Start R_<Config_ICU>_Software_Stop R_<Config_ICU>_SoftwareInterrupt_Generate R_<Config_ICU>_Software2_Start R_<Config_ICU>_Software2_Stop R_<Config_ICU>_SoftwareInterrupt2_Generate
	<Config_ICU>_user.c	R_<Config_ICU>_Create_UserInit r_<Config_ICU>_irqn_interrupt r_<Config_ICU>_software_interrupt r_<Config_ICU>_software2_interrupt r_<Config_ICU>_nmi_interrupt
	<Config_ICU>.h	―
消費電力低減機能	<Config_LPC>.c	R_<Config_LPC>_Create R_<Config_LPC>_AllModuleClockStop R_<Config_LPC>_Sleep R_<Config_LPC>_DeepSleep R_<Config_LPC>_SoftwareStandby R_<Config_LPC>_DeepSoftwareStandby R_<Config_LPC>_ChangeOperatingPowerControl R_<Config_LPC>_ChangeSleepModeReturnClock R_<Config_LPC>_SetVOLSR_PGAVLS
	<Config_LPC>_user.c	R_<Config_LPC>_Create_UserInit
	<Config_LPC>.h	―
ローパワータイマ	<Config_LPT>.c	R_<Config_LPT>_Create R_<Config_LPT>_Start R_<Config_LPT>_Stop
	<Config_LPT>_user.c	R_<Config_LPT>_Create_UserInit
	<Config_LPT>.h	―
ノーマルモード	<Config_MTU0>.c	R_<Config_MTU0>_Create R_<Config_MTU0>_Start R_<Config_MTU0>_Stop
	<Config_MTU0>_user.c	R_<Config_MTU0>_Create_UserInit r_<Config_MTU0>_tgimn_interrupt r_<Config_MTU0>_tginm_interrupt r_<Config_MTU0>_tcivn_interrupt r_<Config_MTU0>_tcinv_interrupt
	<Config_MTU0>.h	―

表 2.8 出力ファイル(8/13)

周辺機能	ファイル名	API 関数名
位相計数モードタイマ	<Config_MTU1>.c	R_<Config_MTU1>_Create R_<Config_MTU1>_Start R_<Config_MTU1>_Stop R_<Config_MTU1_MTU2>_MTU_Start R_<Config_MTU1_MTU2>_MTU_Stop
	<Config_MTU1>_user.c	R_<Config_MTU1>_Create_UserInit r_<Config_MTU1>_tgimn_interrupt r_<Config_MTU1>_tginm_interrupt r_<Config_MTU1>_tcivn_interrupt r_<Config_MTU1>_tcinv_interrupt r_<Config_MTU1>_tciun_interrupt r_<Config_MTU1>_tcinu_interrupt
	<Config_MTU1>.h	－
ポートアウトプットイネーブル	<Config_POE>.c	R_<Config_POE>_Create R_<Config_POE>_Start R_<Config_POE>_Stop R_<Config_POE>_Set_HiZ_MTUn R_<Config_POE>_Clear_HiZ_MTUn R_<Config_POE>_Set_HiZ_GPTn R_<Config_POE>_Clear_HiZ_GPTn
	<Config_POE>_user.c	R_<Config_POE>_Create_UserInit r_<Config_POE>_oein_interrupt
	<Config_POE>.h	－
ポート	<Config_PORT>.c	R_<Config_PORT>_Create
	<Config_PORT>_user.c	R_<Config_PORT>_Create_UserInit
	<Config_PORT>.h	－
プログラマブルパルスジェネレータ	<Config_PPG0>.c	R_<Config_PPG0>_Create
	<Config_PPG0>_user.c	R_<Config_PPG0>_Create_UserInit
	<Config_PPG0>.h	－
PWM モードタイマ	<Config_MTU0>.c	R_<Config_MTU0>_Create R_<Config_MTU0>_Start R_<Config_MTU0>_Stop
	<Config_MTU0>_user.c	R_<Config_MTU0>_Create_UserInit r_<Config_MTU0>_tgimn_interrupt r_<Config_MTU0>_tginm_interrupt r_<Config_MTU0>_tcivn_interrupt r_<Config_MTU0>_tcinv_interrupt
	<Config_MTU0>.h	－

表 2.9 出力ファイル(9/13)

周辺機能	ファイル名	API 関数名
リアルタイムクロック	<Config_RTC>.c	R_<Config_RTC>_Create R_<Config_RTC>_Start R_<Config_RTC>_Stop R_<Config_RTC>_Restart R_<Config_RTC>_Restart_BinaryCounter R_<Config_RTC>_Set_CalendarCounterValue R_<Config_RTC>_Get_CalendarCounterValue R_<Config_RTC>_Get_CalendarTimeCaptureValuen R_<Config_RTC>_Set_BinaryCounterValue R_<Config_RTC>_Get_BinaryCounterValue R_<Config_RTC>_Get_BinaryTimeCaptureValuen R_<Config_RTC>_Set_RTCOUTOn R_<Config_RTC>_Set_RTCOUTOff R_<Config_RTC>_Set_CalendarAlarm R_<Config_RTC>_Set_BinaryAlarm R_<Config_RTC>_Set_ConstPeriodInterruptOn R_<Config_RTC>_Set_ConstPeriodInterruptOff R_<Config_RTC>_Set_CarryInterruptOn R_<Config_RTC>_Set_CarryInterruptOff R_<Config_RTC>_Enable_Alarm_Interruptf R_<Config_RTC>_Disable_Alarm_Interrupt
	<Config_RTC>_user.c	R_<Config_RTC>_Create_UserInit r_<Config_RTC>_alm_interrupt r_<Config_RTC>_prd_interrupt r_<Config_RTC>_cup_interrupt
	<Config_RTC>.h	ー
リモコン信号受信機能	<Config_REMC0>.c	R_<Config_REMC0>_Create R_<Config_REMC0>_Start R_<Config_REMC0>_Stop R_<Config_REMC0>_Read
	<Config_REMC0>_user.c	R_<Config_REMC0>_Create_UserInit r_<Config_REMC0>_remcin_interrupt r_<Config_REMC0>_callback_comparematch r_<Config_REMC0>_callback_receiveerror r_<Config_REMC0>_callback_receiveend r_<Config_REMC0>_callback_bufferfull r_<Config_REMC0>_callback_header r_<Config_REMC0>_callback_data0 r_<Config_REMC0>_callback_data1 r_<Config_REMC0>_callback_specialdata
	<Config_REMC0>.h	ー

表 2.10 出力ファイル(10/13)

周辺機能	ファイル名	API 関数名
SCI/SCIF 調歩同期式モード	<Config_SCI0>.c	R_<Config_SCI0>_Create R_<Config_SCI0>_Start R_<Config_SCI0>_Stop R_<Config_SCI0>_Serial_Send R_<Config_SCI0>_Serial_Receive R_<Config_SCI0>_Serial_Multiprocessor_Send R_<Config_SCI0>_Serial_Multiprocessor_Receive
	<Config_SCI0>_user.c	R_<Config_SCI0>_Create_UserInit r_<Config_SCI0>_transmitend_interrupt r_<Config_SCI0>_transmit_interrupt r_<Config_SCI0>_receive_interrupt r_<Config_SCI0>_receiveerror_interrupt r_<Config_SCI0>_teif_interrupt r_<Config_SCI0>_txif_interrupt r_<Config_SCI0>_rxif_interrupt r_<Config_SCI0>_drif_interrupt r_<Config_SCI0>_erif_interrupt r_<Config_SCI0>_brif_interrupt r_<Config_SCI0>_callback_transmitend r_<Config_SCI0>_callback_receiveend r_<Config_SCI0>_callback_receiveerror r_<Config_SCI0>_callback_error
	<Config_SCI0>.h	ー
SCI/SCIF クロック同期式モード	<Config_SCI0>.c	R_<Config_SCI0>_Create R_<Config_SCI0>_Start R_<Config_SCI0>_Stop R_<Config_SCI0>_Serial_Send R_<Config_SCI0>_Serial_Receive R_<Config_SCI0>_Serial_Send_Receive
	<Config_SCI0>_user.c	R_<Config_SCI0>_Create_UserInit r_<Config_SCI0>_transmitend_interrupt r_<Config_SCI0>_transmit_interrupt r_<Config_SCI0>_receive_interrupt r_<Config_SCI0>_receiveerror_interrupt r_<Config_SCI0>_teif_interrupt r_<Config_SCI0>_txif_interrupt r_<Config_SCI0>_rxif_interrupt r_<Config_SCI0>_erif_interrupt r_<Config_SCI0>_brif_interrupt r_<Config_SCI0>_callback_transmitend r_<Config_SCI0>_callback_receiveend r_<Config_SCI0>_callback_receiveerror r_<Config_SCI0>_callback_error
	<Config_SCI0>.h	ー

表 2.11 出力ファイル(11/13)

周辺機能	ファイル名	API 関数名
シングルスキャンモード S12AD	<Config_S12AD0>.c	R_<Config_S12AD0>_Create R_<Config_S12AD0>_Start R_<Config_S12AD0>_Stop R_<Config_S12AD0>_Get_ValueResult R_<Config_S12AD0>_Set_CompareValue R_<Config_S12AD0>_Set_CompareAValue R_<Config_S12AD0>_Set_CompareBValue
	<Config_S12AD0>_user.c	R_<Config_S12AD0>_Create_UserInit r_<Config_S12AD0>_interrupt r_<Config_S12AD0>_compare_interrupt r_<Config_S12AD0>_compare_interruptA r_<Config_S12AD0>_compare_interruptB
	<Config_S12AD0>.h	―
スマートカードインタフェース	<Config_SCI0>.c	R_<Config_SCI0>_Create R_<Config_SCI0>_Start R_<Config_SCI0>_Stop R_<Config_SCI0>_SmartCard_Send R_<Config_SCI0>_SmartCard_Receive
	<Config_SCI0>_user.c	R_<Config_SCI0>_Create_UserInit r_<Config_SCI0>_transmit_interrupt r_<Config_SCI0>_receive_interrupt r_<Config_SCI0>_receiveerror_interrupt r_<Config_SCI0>_callback_transmitend r_<Config_SCI0>_callback_receiveend r_<Config_SCI0>_callback_receiveerror
	<Config_SCI0>.h	―
SPI クロック同期式モード	<Config_RSPI0>.c	R_<Config_RSPI0>_Create R_<Config_RSPI0>_Start R_<Config_RSPI0>_Stop R_<Config_RSPI0>_Send R_<Config_RSPI0>_Send_Receive R_<Config_RSPI0>_SPI_Master_Send R_<Config_RSPI0>_SPI_Master_Send_Receive R_<Config_RSPI0>_SPI_Slave_Send R_<Config_RSPI0>_SPI_Slave_Send_Receive
	<Config_RSPI0>_user.c	R_<Config_RSPI0>_Create_UserInit r_<Config_RSPI0>_receive_interrupt r_<Config_RSPI0>_transmit_interrupt r_<Config_RSPI0>_error_interrupt r_<Config_RSPI0>_idle_interrupt r_<Config_RSPI0>_transmitend_interrupt r_<Config_RSPI0>_receiveerror_interrupt r_<Config_RSPI0>_callback_receiveend r_<Config_RSPI0>_callback_transmitend r_<Config_RSPI0>_callback_error
	<Config_RSPI0>.h	―

表 2.12 出力ファイル(12/13)

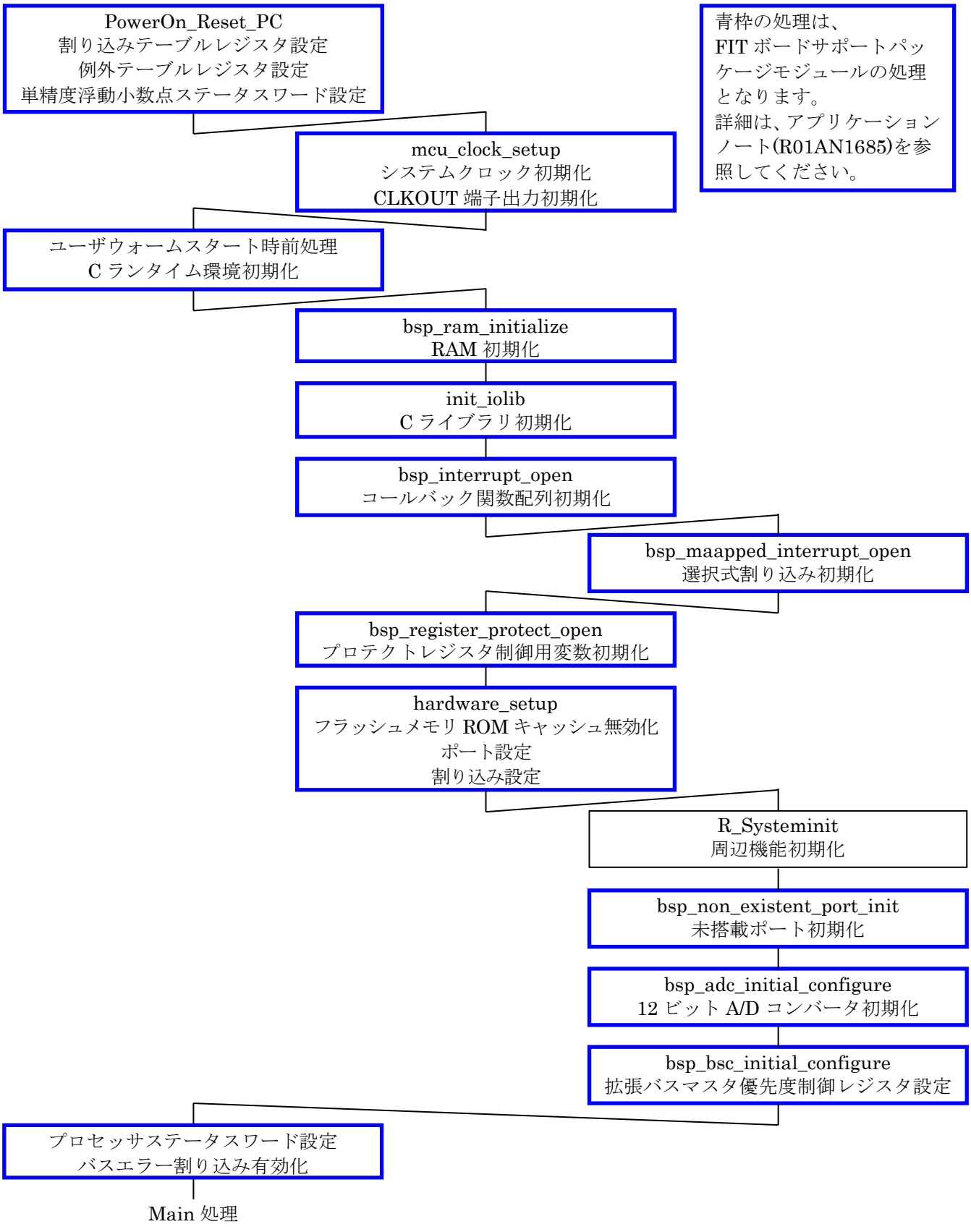
周辺機能	ファイル名	API 関数名
SPI 動作モード	<Config_RSPI0>.c	R_<Config_RSPI0>_Create R_<Config_RSPI0>_Start R_<Config_RSPI0>_Stop R_<Config_RSPI0>_Send R_<Config_RSPI0>_Send_Receive
	<Config_RSPI0>_user.c	R_<Config_RSPI0>_Create_UserInit r_<Config_RSPI0>_receive_interrupt r_<Config_RSPI0>_transmit_interrupt r_<Config_RSPI0>_error_interrupt r_<Config_RSPI0>_idle_interrupt r_<Config_RSPI0>_callback_receiveend r_<Config_RSPI0>_callback_transmitend r_<Config_RSPI0>_callback_error
	<Config_RSPI0>.h	－
電圧検出回路	<Config_LVD1>.c	R_<Config_LVD1>_Create R_<Config_LVD1>_Start R_<Config_LVD1>_Stop
	<Config_LVD1>_user.c	R_<Config_LVD1>_Create_UserInit r_<Config_LVD1>_lvdn_interrupt
	<Config_LVD1>.h	－
ウォッチドッグタイマ	<Config_WDT>.c	R_<Config_WDT>_Create R_<Config_WDT>_Restart
	<Config_WDT>_user.c	R_<Config_WDT>_Create_UserInit r_<Config_WDT>_wuni_interrupt r_<Config_WDT>_iwuni_interrupt r_<Config_WDT>_nmi_interrupt
	<Config_WDT>.h	－
連続スキャンモード DSAD	<Config_DSAD0>.c	R_<Config_DSAD0>_Create R_<Config_DSAD0>_Start R_<Config_DSAD0>_Stop R_<Config_DSAD0>_Set_SoftwareTrigger R_<Config_DSAD0>_Get_ValueResult R_<Config_DSAD0>_Set_Chm_DisconnectDetection
	<Config_DSAD0>_user.c	R_<Config_DSAD0>_Create_UserInit r_<Config_DSAD0>_adin_interrupt r_<Config_DSAD0>_scanendn_interrupt
	<Config_DSAD0>.h	－
シングルスキャンモード DSAD	<Config_DSAD0>.c	R_<Config_DSAD0>_Create R_<Config_DSAD0>_Start R_<Config_DSAD0>_Stop R_<Config_DSAD0>_Set_SoftwareTrigger R_<Config_DSAD0>_Get_ValueResult R_<Config_DSAD0>_Set_Chm_DisconnectDetection
	<Config_DSAD0>_user.c	R_<Config_DSAD0>_Create_UserInit r_<Config_DSAD0>_adin_interrupt r_<Config_DSAD0>_scanendn_interrupt
	<Config_DSAD0>.h	－

表 2.13 出力ファイル(13/13)

周辺機能	ファイル名	API 関数名
Δ-Σ モジュールインタフェース	<Config_DSMIF0>.c	R_<Config_DSMIF0>_Create R_<Config_DSMIF0>_Start R_<Config_DSMIF0>_Stop
	<Config_DSMIF0>_user.c	R_<Config_DSMIF0>_Create_UserInit r_<Config_DSMIF0>_ocdin_interrupt r_<Config_DSMIF0>_scdin_interrupt r_<Config_DSMIF0>_sumein_interrupt
	<Config_DSMIF0>.h	―
アナログフロントエンド	<Config_AFE>.c	R_<Config_AFE>_Create
	<Config_AFE>_user.c	R_<Config_AFE>_Create_UserInit
	<Config_AFE>.h	―
モータ (相補 PWM モードタイマ，シングルスキャンモード S12AD)	<Config_MTU3_MTU4>.c	R_<Config_MTU3_MTU4>_Create R_<Config_MTU3_MTU4>_StartTimerCount R_<Config_MTU3_MTU4>_StopTimerCount R_<Config_MTU3_MTU4>_StartTimerCtrl R_<Config_MTU3_MTU4>_StopTimerCtrl R_<Config_MTU3_MTU4>_UpdDuty R_<Config_MTU3_MTU4>_StartAD R_<Config_MTU3_MTU4>_StopAD R_<Config_MTU3_MTU4>_AdcGetConvVal
	<Config_MTU3_MTU4>_user.c	R_<Config_MTU3_MTU4>_Create_UserInit r_<Config_MTU3_MTU4>_CrestInterrupt r_<Config_MTU3_MTU4>_ad_interrupt
	<Config_MTU3_MTU4>.h	―
LCD コントローラ	<Config_LCD>.c	R_<Config_LCD>_Create R_<Config_LCD>_Start R_<Config_LCD>_Stop R_<Config_LCD>_Voltage_On R_<Config_LCD>_Voltage_Off
	<Config_LCD>_user.c	R_<Config_LCD>_Create_UserInit
	<Config_LCD>.h	―

3. 初期化

本章では、コード生成が出力するファイルによる初期化の流れについて説明します。



4. API 関数

本章では、コード生成が出力する API 関数について説明します。

4.1 概要

以下に、コード生成が API 関数を出力する際の命名規則を示します。

- グローバル関数名
  先頭に “R” を付与し、構成単語の先頭のみ大文字。
  ただし、ユーザが変更可能な<>内は、大文字、小文字ともに使用できます。
  <>内に設定されるデフォルトの名前は、”Config”と周辺機能名、チャネル番号を付与します。

- スタティック関数名
  先頭に “r” を付与し、全て小文字。
  ただし、ユーザが変更可能な<>内は、大文字、小文字ともに使用できます。
  <>内に設定されるデフォルトの名前は、”Config”と周辺機能名、チャネル番号を付与します。

- マクロ名
  すべて大文字。
  なお、先頭に “ 数字 ” が付与されている場合、該当数字（16 進数値）とマクロ値は同値。

- ローカル変数名
  すべて小文字。

- グローバル変数名
  先頭に “g” を付与し、構成単語の先頭のみ大文字。

- グローバル変数へのポインタ名
  先頭に “gp” を付与し、構成単語の先頭のみ大文字。

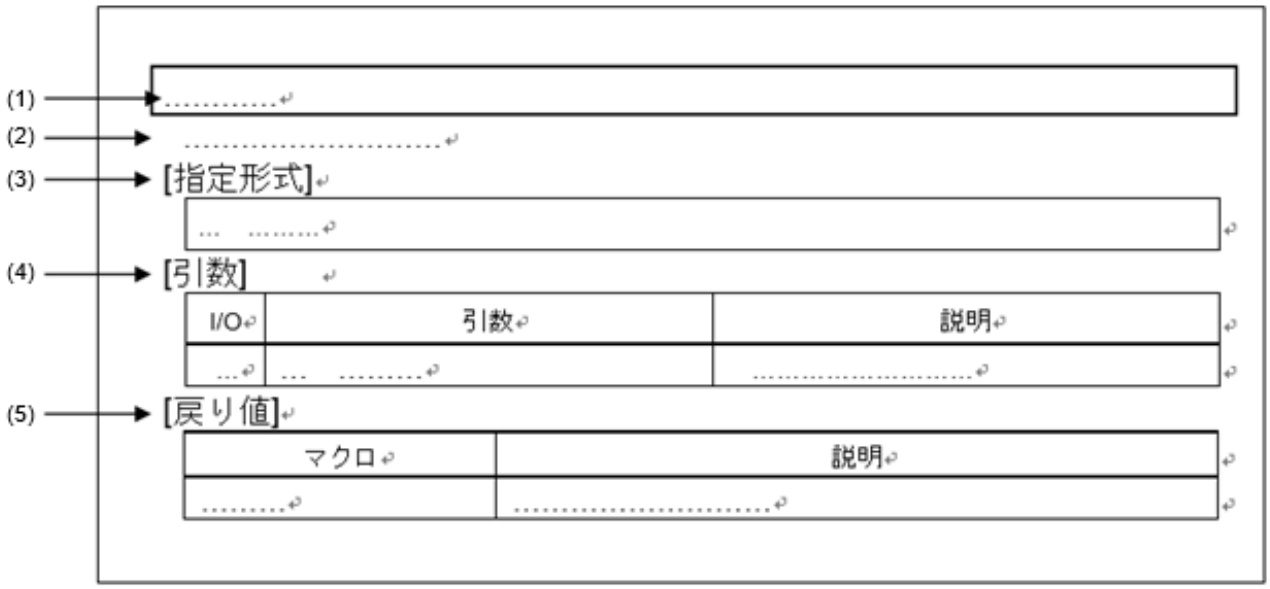
- 列挙指定子 enum の要素名
  すべて大文字。

備考　スマート・コンフィグレータの生成するコードには、レジスタの反映待ち処理等でfor 文、while 文、do while 文（ループ処理）を使用しています。
無限ループに対するフェールセーフ処理が必要な場合は、生成されたコードを確認の上、処理を追加してください。

4.2 関数リファレンス

本節では、コード生成が出力する API 関数について、次の記述フォーマットに従って説明します。

図 4.1 API 関数の記述フォーマット



(2) 機能

API 関数の機能概要を示しています。

(3) ［指定形式］

API 関数を C 言語で呼び出す際の記述形式を示しています。

(4) ［引数］

API 関数の引数を次の形式で示しています。

I/O	引数	説明
(a)	(b)	(c)

(a) I/O

引数の種類

I … 入力引数

O … 出力引数

(b) 引数

引数のデータタイプ

(c) 説明

引数の説明

(5) ［戻り値］

API 関数からの戻り値を次の形式で示しています。

マクロ	説明
(a)	(b)

(a) マクロ

戻り値のマクロ

(b) 説明

戻り値の説明

4.2.1 共通

以下に、コード生成ツールが共通用として出力する API 関数の一覧を示します。

表 4.1 共通用 API 関数

API 関数名	機能概要
r_undefined_exception	未定義命令例外の発生に伴う処理を行います。
PowerON_Reset	リセットの発生に伴う処理を行います。
hardware_setup	各種ハードウエアを制御するうえで必要となる初期化処理を行います。
R_Systeminit	各種周辺機能を制御するうえで必要となる初期化処理を行います。
R_CGC_Create	クロック発生回路を制御するうえで必要となる初期化処理を行います。
R_CGC_Create_UserInit	クロック発生回路に関するユーザ独自の初期化処理を行います。
R_Interrupt_Create	「割り込み」タブから設定したグループ割り込みや高速割り込みの設定を行います。
R_Pins_Create	「端子」タブから設定したマルチファンクションピンコントローラへの設定を行います。
main	main 関数です。

r_undefined_exception

未定義命令例外の発生に伴う処理を行います。

備考　本 API 関数は、未定義命令（実装されていない命令）の実行を検出した際に発生する未定義命令例外に対応した割り込み処理として呼び出されます。

[指定形式]

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

[引数]

なし

[戻り値]

なし

PowerON_Reset

リセットの発生に伴う処理を行います。

備考　本 API 関数は、パワーオンリセット回路による内部リセットに対応した割り込み処理として呼び出されます。

[指定形式]

void	PowerON_Reset (void);

[引数]

なし

[戻り値]

なし

hardware_setup

各種ハードウエアを制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、PowerON_Reset のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void    hardware_setup ( void );
```

[引数]

なし

[戻り値]

なし

R_Systeminit

各種周辺機能を制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、hardware_setup のコールバック・ルーチンとして呼び出されます。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_CGC_Create

クロック発生回路を制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_CGC_Create_UserInit

クロック発生回路に関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_CGC_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_Interrupt_Create

グループ割り込みや高速割り込みの設定を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_Pins_Create

マルチファンクションピンコントローラへの設定を行います。

[指定形式]

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

[引数]

なし

[戻り値]

なし

main

main 関数です。

備考 本 API 関数は、PowerON_Reset のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void    main ( void );
```

[引数]

なし

[戻り値]

なし

4.2.2 8 ビットタイマ

以下に、コード生成ツールが 8 ビットタイマ用として出力する API 関数の一覧を示します。

表 4.2 8 ビットタイマ用 API 関数

API 関数名	機能概要
R_<Config_TMR0>_Create	8 ビットタイマを制御するうえで必要となる初期化処理を行います。
R_<Config_TMR0>_Start	カウントを開始します。
R_<Config_TMR0>_Stop	カウントを終了します。
R_<Config_TMR0>_Create_UserInit	8 ビットタイマに関するユーザ独自の初期化処理を行います。
r_<Config_TMR0>_cmimn_interrupt	コンペアマッチ割り込みの発生に伴う処理を行います。
r_<Config_TMR0>_ovin_interrupt	オーバフロー割り込みの発生に伴う処理を行います。

R_<Config_TMR0>_Create

8 ビットタイマを制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void    R_<Config_TMR0>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_TMR0>_Start

カウントを開始します。

[指定形式]

```
void        R_<Config_TMR0>_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_TMR0>_Stop

カウントを終了します。

[指定形式]

```
void        R_<Config_TMR0>_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_TMR0>_Create_UserInit

8 ビットタイマに関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_TMR0>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_TMR0>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

r_<Config_TMR0>_cmi*mn*_interrupt

コンペアマッチ割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_TMR0>_cmi*mn*_interrupt (void);

備考　n はチャネル番号を、m はタイマコンスタントレジスタ番号を意味します。

[引数]

なし

[戻り値]

なし

r_<Config_TMR0>_ovi*n*_interrupt

オーバフロー割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_TMR0>_ovi*n*_interrupt (void);

備考 *n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

使用例

ワンショットタイマとして使用する

main.c

```
#include "r_smc_entry.h"
void main(void)
{
    /* Start TMR channel 0 counter */
    R_Config_TMR0_Start();

    while (1U)
    {
        nop();
    }
}
```

Config_TMR0_user.c

```
static void r_Config_TMR0_cmia0_interrupt(void)
{
    /* Start user code for r_Config_TMR0_cmia0_interrupt. Do not edit comment generated here */
    /* Stop TMR channel 0 counter */
    R_Config_TMR0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

4.2.3 バス

以下に、コード生成ツールがバス用として出力する API 関数の一覧を示します。

表 4.3 バス用 API 関数

API 関数名	機能概要
R_<Config_BSC>_Create	バスを制御するうえで必要となる初期化処理を行います。
R_<Config_BSC>_Error_Monitoring_Start	バスエラー（不正アドレスアクセス）の検出を許可します。
R_<Config_BSC>_Error_Monitoring_Stop	バスエラー（不正アドレスアクセス）の検出を禁止します。
R_<Config_BSC>_InitializeSDRAM	SDRAM コントローラの初期化を行います。
R_<Config_BSC>_Create_UserInit	バスに関するユーザ独自の初期化処理を行います。
r_<Config_BSC>_buserr_interrupt	バスエラー（不正アドレスアクセス）の発生に伴う処理を行います。

R_<Config_BSC>_Create

バスを制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_<Config_BSC>_Error_Monitoring_Start

バスエラー（不正アドレスアクセス）の検出を許可します。

[指定形式]

```
void        R_<Config_BSC>_Error_Monitoring_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_BSC>_Error_Monitoring_Stop

バスエラー（不正アドレスアクセス）の検出を禁止します。

[指定形式]

```
void        R_<Config_BSC>_Error_Monitoring_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_BSC>_InitializeSDRAM

SDRAM コントローラの初期化を行います。

[指定形式]

```
void        R_<Config_BSC>_InitializeSDRAM ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_BSC>_Create_UserInit

バスに関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_BSC>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_BSC>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

r_<Config_BSC>_buserr_interrupt

バスエラー（不正アドレスアクセス）の発生に伴う処理を行います。

備考 1. 本 API 関数は、処理プログラムが不正なアドレス領域にアクセスした場合に発生するバスエラー（不正アドレスアクセス）に対応した割り込み処理として呼び出されます。

備考 2. 本 API 関数内でバスエラーステータスレジスタ 1（BERSR1）の MST ビットを読み出すことにより、バスエラーの発生要因となったバスマスタを確認することができます。

備考 3. 本 API 関数内でバスエラーステータスレジスタ 2（BERSR2）の ADDR ビットを読み出すことにより、バスエラーの発生要因となった不正アドレス（上位 13 ビット）を確認することができます。

[指定形式]

```
void        r_<Config_BSC>_buserr_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

使用例

バスエラーが発生したアクセスのアドレスを取得する

main.c

```
#include "r_smc_entry.h"
void main(void)
{
    /* Enable BUSERR interrupt in ICU */
    R_Config_BSC_Error_Monitoring_Start();

    while (1U)
    {
        nop();
    }
}
```

Config_BSC_user.c

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

void r_Config_BSC_buserr_interrupt(void)
{
    /* Start user code for r_Config_BSC_buserr_interrupt. Do not edit comment generated here */
    /* Restore an address that was accessed when a bus error occurred */
    if (1U == BSC.BERSR1.BIT.IA)
    {
        g_bsc_buserr_addr = ((uint16_t)(BSC.BERSR2.WORD)>>3U);
    }

    /* Clear the bus error status registers */
    BSC.BERCLR.BIT.STSCLR = 1U;
    /* End user code. Do not edit comment generated here */
}
```

4.2.4 クロック周波数精度測定回路

以下に、コード生成ツールがクロック周波数精度測定回路用として出力する API 関数の一覧を示します。

表 4.4 クロック周波数精度測定回路用 API 関数

API 関数名	機能概要
R_<Config_CAC>_Create	クロック周波数精度測定回路を制御するうえで必要となる初期化処理を行います。
R_<Config_CAC>_Start	クロック周波数の精度測定を開始します。
R_<Config_CAC>_Stop	クロック周波数の精度測定を終了します。
R_<Config_CAC>_Create_UserInit	クロック周波数精度測定回路に関するユーザ独自の初期化処理を行います。
r_<Config_CAC>_mendf_interrupt	測定終了割り込みの発生に伴う処理を行います。 （デバイスグループにより、API 関数名が異なります。）
r_<Config_CAC>_mendi_interrupt	
r_<Config_CAC>_ferrf_interrupt	周波数エラー割り込みの発生に伴う処理を行います。 （デバイスグループにより、API 関数名が異なります。）
r_<Config_CAC>_ferri_interrupt	
r_<Config_CAC>_ovff_interrupt	オーバフロー割り込みの発生に伴う処理を行います。 （デバイスグループにより、API 関数名が異なります。）
r_<Config_CAC>_ovfi_interrupt	

R_<Config_CAC>_Create

クロック周波数精度測定回路（CAC）を制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_<Config_CAC>_Start

クロック周波数の精度測定を開始します。

備考　本 API 関数を実行する前に、CAC ステータスレジスタ(CASTR)をクリアしてください。
フラグが立ったままの場合、本関数実行と同時に割り込みが発生することがあります。

[指定形式]

void	R_<Config_CAC>_Start (void);

[引数]

なし

[戻り値]

なし

R_<Config_CAC>_Stop

クロック周波数の精度測定を終了します。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_<Config_CAC>_Create_UserInit

クロック周波数精度測定回路（CAC）に関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_CAC>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_CAC>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

r_<Config_CAC>_mendf_interrupt

r_<Config_CAC>_mendi_interrupt

測定終了割り込みの発生に伴う処理を行います。

備考 本 API 関数は、クロック周波数精度測定回路が基準信号の有効エッジを検出した場合に発生する測定終了割り込みに対応した割り込み処理として呼び出されます。

[指定形式]

void	r_<Config_CAC>_mendf_interrupt (void);

void	r_<Config_CAC>_mendi_interrupt (void);

備考 デバイスグループにより、API 関数名が異なります。

[引数]

なし

[戻り値]

なし

r_<Config_CAC>_ferrf_interrupt

r_<Config_CAC>_ferri_interrupt

周波数エラー割り込みの発生に伴う処理を行います。

備考　本 API 関数は、クロック周波数が有効範囲（下限値から上限値まで）を外れた場合に発生する周波数エラー割り込みに対応した割り込み処理として呼び出されます。

[指定形式]

```
void    r_<Config_CAC>_ferrf_interrupt ( void );
```

```
void    r_<Config_CAC>_ferri_interrupt ( void );
```

備考　デバイスグループにより、API 関数名が異なります。

[引数]

なし

[戻り値]

なし

r_<Config_CAC>_ovff_interrupt

r_<Config_CAC>_ovfi_interrupt

オーバフロー割り込みの発生に伴う処理を行います。

備考　本 API 関数は、カウンタがオーバフローした場合に発生するオーバフロー割り込みに対応した割り込み処理として呼び出されます。

[指定形式]

void	r_<Config_CAC>_ovff_interrupt (void);

void	r_<Config_CAC>_ovfi_interrupt (void);

備考　デバイスグループにより、API 関数名が異なります。

[引数]

なし

[戻り値]

なし

使用例

周波数エラーをカウントする

main.c

```
#include "r_smc_entry.h"
void main(void)
{
    /* Clear interrupt request flag */
    CAC.CAICR.BIT.FERRFCL = 1U;
    CAC.CAICR.BIT.MENDFCL = 1U;
    CAC.CAICR.BIT.OVFFCL = 1U;

    /* Enable clock frequency measurement */
    R_Config_CAC_Start();

    while (1U)
    {
        nop();
    }
}
```

Config_CAC_user.c

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

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

void r_Config_CAC_ferri_interrupt(void)
{
    /* Start user code for r_Config_CAC_ferri_interrupt. Do not edit comment generated here */
    /* Add the error countor */
    g_cac_ferri_cnt++;
    /* End user code. Do not edit comment generated here */
}
```

4.2.5 コンパレータ

以下に、コード生成ツールがコンパレータ用として出力する API 関数の一覧を示します。

表 4.5 コンパレータ用 API 関数

API 関数名	機能概要
R_<Config_CMPB0>_Create	コンパレータを制御するうえで必要となる初期化処理を行います。
R_<Config_CMPB0>_Start	アナログ入力電圧の比較を開始します。
R_<Config_CMPB0>_Stop	アナログ入力電圧の比較を終了します。
R_<Config_CMPB0>_Create_UserInit	コンパレータに関するユーザ独自の初期化処理を行います。
r_<Config_CMPB0>_cmpbn_interrupt	コンパレータ B*n* 割り込みの発生に伴う処理を行います。

R_<Config_CMPB0>_Create

コンパレータを制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void    R_<Config_CMPB0>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_CMPB0>_Start

アナログ入力電圧の比較を開始します。

[指定形式]

```
void        R_<Config_CMPB0>_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_CMPB0>_Stop

アナログ入力電圧の比較を終了します。

[指定形式]

```
void        R_<Config_CMPB0>_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_CMPB0>_Create_UserInit

コンパレータに関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_CMPB0>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_CMPB0>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

r_<Config_CMPB0>_cmpb*n*_interrupt

コンパレータ B*n* 割り込みの発生に伴う処理を行います。

備考　本 API 関数は、アナログ入力電圧とリファレンス入力電圧の比較結果が変化した場合に発生するコンパレータ B*n* 割り込みに対応した割り込み処理として呼び出されます。

[指定形式]

void	r_<Config_CMPB0>_cmpb*n*_interrupt (void);

備考　*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

使用例

比較結果の変化でフラグを立てる

main.c

```
#include "r_smc_entry.h"
void main(void)
{
    /* Start comparator B0 */
    R_Config_CMPB0_Start();

    while (1U)
    {
        nop();
    }
}
```

Config_CMPB0_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_cmpb0_f;
/* End user code. Do not edit comment generated here */

void R_Config_CMPB0_Create_UserInit(void)
{
    /* Start user code for user init. Do not edit comment generated here */
    /* Clear the flag */
    g_cmpb0_f = 0U;
    /* End user code. Do not edit comment generated here */
}

static void r_Config_CMPB0_cmpb0_interrupt(void)
{
    /* Start user code for r_Config_CMPB0_cmpb0_interrupt. Do not edit comment generated here
*/
    /* Set the flag */
    g_cmpb0_f = 1U;
    /* End user code. Do not edit comment generated here */
}
```

4.2.6 コンペアマッチタイマ

以下に、コード生成ツールがコンペアマッチタイマ（CMT/CMTW）用として出力する API 関数の一覧を示します。

表 4.6 コンペアマッチタイマ用 API 関数

API 関数名	機能概要
R_<Config_CMT0>_Create	コンペアマッチタイマ（CMT/CMTW）を制御するうえで必要となる初期化処理を行います。
R_<Config_CMT0>_Start	カウントを開始します。
R_<Config_CMT0>_Stop	カウントを終了します。
R_<Config_CMT0>_Create_UserInit	コンペアマッチタイマ（CMT/CMTW）に関するユーザ独自の初期化処理を行います。
r_<Config_CMT0>_cmin_interrupt	コンペアマッチ割り込み（CMI*n*）の発生に伴う処理を行います。
r_<Config_CMT0>_cmwin_interrupt	コンペアマッチ割り込み（CMWI*n*）の発生に伴う処理を行います。
r_<Config_CMT0>_icmin_interrupt	インプットキャプチャ割り込みの発生に伴う処理を行います。
r_<Config_CMT0>_ocmin_interrupt	アウトプットコンペア割り込みの発生に伴う処理を行います。

R_<Config_CMT0>_Create

コンペアマッチタイマ（CMT/CMTW）を制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void    R_<Config_CMT0>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_CMT0>_Start

カウントを開始します。

[指定形式]

void	R_<Config_CMT0>_Start (void);

[引数]

なし

[戻り値]

なし

R_<Config_CMT0>_Stop

カウントを終了します。

[指定形式]

```
void        R_<Config_CMT0>_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_CMT0>_Create_UserInit

コンペアマッチタイマ（CMT/CMTW）に関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_CMT0>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_CMT0>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

r_<Config_CMT0>_cmi*n*_interrupt

コンペアマッチ割り込み（CMI*n*）の発生に伴う処理を行います。

備考　本 API 関数は、現在カウント値“ コンペアマッチタイマカウンタ（CMCNT）の値 ”と規定カウント値“ コンペアマッチタイマコンスタントレジスタ（CMCOR）の値 ”が一致した場合に発生するコンペアマッチ割り込み（CMI*n*）に対応した割り込み処理として呼び出されます。

[指定形式]

void	r_<Config_CMT0>_cmi*n*_interrupt (void);

備考　*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

r_<Config_CMT0>_cmwi*n*_interrupt

コンペアマッチ割り込み（CMWI*n*）の発生に伴う処理を行います。

備考　本 API 関数は、現在カウント値“ コンペアマッチタイマカウンタ（CMWCNT）の値 ”と規定カウント値“ コンペアマッチタイマコンスタントレジスタ（CMWCOR）の値 ”が一致した場合に発生するコンペアマッチ割り込み（CMWI*n*）に対応した割り込み処理として呼び出されます。

[指定形式]

```
void    r_<Config_CMT0>_cmwin_interrupt ( void );
```

備考　*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

r_<Config_CMT0>_icmi*n*_interrupt

インプットキャプチャ割り込み の発生に伴う処理を行います。

[指定形式]

void	r_<Config_CMT0>_icmi*n*_interrupt (void);

備考 *n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

r_<Config_CMT0>_ocmi*n*_interrupt

アウトプットコンペア割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_CMT0>_ocmi*n*_interrupt (void);

備考 *n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

使用例

ワンショットタイマとして使用する

main.c

```
#include "r_smc_entry.h"
void main(void)
{
    /* Start CMT channel 0 counter */
    R_Config_CMT0_Start();

    while (1U)
    {
        nop();
    }
}
```

Config_CMT0_user.c

```
static void r_Config_CMT0_cmi0_interrupt(void)
{
    /* Start user code for r_Config_CMT0_cmi0_interrupt. Do not edit comment generated here */
    /* Stop CMT channel 0 counter */
    R_Config_CMT0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

4.2.7 相補 PWM モードタイマ

以下に、コード生成ツールが相補 PWM モードタイマ用として出力する API 関数の一覧を示します。

表 4.7 相補 PWM モードタイマ用 API 関数

API 関数名	機能概要
R_<Config_MTU3_MTU4>_Create	相補 PWM モードタイマを制御するうえで必要となる初期化処理を行います。
R_<Config_MTU3_MTU4>_Start	カウンタを開始します。
R_<Config_MTU3_MTU4>_Stop	カウンタを終了します。
R_<Config_MTU3_MTU4>_Create_UserInit	相補 PWM モードタイマに関するユーザ独自の初期化処理を行います。
r_<Config_MTU3_MTU4>_tgimn_interrupt	コンペアマッチ割り込みの発生に伴う処理を行います。
r_<Config_MTU3_MTU4>_cj_tgimj_interrupt	コンペアマッチ割り込みの発生に伴う処理を行います。
r_<Config_MTU3_MTU4>_cj_tcivj_interrupt	アンダフロー割り込みの発生に伴う処理を行います。

R_<Config_MTU3_MTU4>_Create

相補 PWM モードタイマを制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void    R_<Config_MTU3_MTU4>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_MTU3_MTU4>_Start

カウンタを開始します。

[指定形式]

```
void        R_<Config_MTU3_MTU4>_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_MTU3_MTU4>_Stop

カウンタを終了します。

[指定形式]

```
void        R_<Config_MTU3_MTU4>_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_MTU3_MTU4>_Create_UserInit

相補 PWM モードタイマに関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_MTU3_MTU4>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_MTU3_MTU4>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

r_<Config_MTU3_MTU4>_tgi*mn*_interrupt

コンペアマッチ割り込みの発生に伴う処理を行います。

備考　本 API 関数は、現在カウント値 “ タイマカウンタ（TCNT）の値 ” と規定カウント値 “ タイマジェネラルレジスタ（TGR）の値 ” が一致した場合に発生するコンペアマッチ割り込みに対応した割り込み処理として呼び出されます。

[指定形式]

void	r_<Config_MTU3_MTU4>_tgi*mn*_interrupt (void);

備考　*n* はチャネル番号を、*m* はタイマジェネラルレジスタ番号を意味します。

[引数]

なし

[戻り値]

なし

r_<Config_MTU3_MTU4>_c*j*_tgi*mj*_interrupt

コンペアマッチ割り込みの発生に伴う処理を行います。

備考　本 API 関数は、現在カウント値 “ タイマカウンタ（TCNT）の値 ” と規定カウント値 “ タイマジェネラルレジスタ（TGR）の値 ” が一致した場合に発生するコンペアマッチ割り込みに対応した割り込み処理として呼び出されます。

[指定形式]

void	r_<Config_MTU3_MTU4>_c*j*_tgi*mj*_interrupt (void);

備考　*j* はチャネル番号を、*m* はタイマジェネラルレジスタ番号を意味します。

[引数]

なし

[戻り値]

なし

r_<Config_MTU3_MTU4>_c*j*_tciv*j*_interrupt

アンダフロー割り込みの発生に伴う処理を行います。

備考　本 API 関数は、タイマカウンタ（TCNT）がアンダフローした場合に発生するアンダフロー割り込みに対応した割り込み処理として呼び出されます。

[指定形式]

void	r_<Config_MTU3_MTU4>_c*j*_tciv*j*_interrupt (void);

備考　*j* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

使用例

U 相の波を上限まで徐々に大きくした後は下限まで徐々に小さくする処理を繰り返す

main.c

```
#include "r_smc_entry.h"
void main(void)
{
    /* Start the MTU6 channel counter */
    R_Config_MTU6_MTU7_Start();

    while (1U)
    {
        nop();
    }
}
```

Config_MTU6_MTU7_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint16_t gu_pulse_u;
volatile int8_t g_pulse_dir_u;
/* End user code. Do not edit comment generated here */

void R_Config_MTU6_MTU7_Create_UserInit(void)
{
    /* Start user code for user init. Do not edit comment generated here */
    gu_pulse_u = _xxxx_6TGRB_VALUE;
    g_pulse_dir_u = 1U;
    /* End user code. Do not edit comment generated here */
}

static void r_Config_MTU6_MTU7_tgib6_interrupt(void)
{
    /* Start user code for r_Config_MTU6_MTU7_tgib6_interrupt. Do not edit comment generated
here */
    gu_pulse_u += g_pulse_dir_u;
    if(gu_pulse_u == _xxxx_TCDRB_VALUE)
    {
        g_pulse_dir_u = -1;
    }
    else if(gu_pulse_u == _xxxx_TDDRB_VALUE)
    {
        g_pulse_dir_u = 1;
    }
    MTU6.TGRB = gu_pulse_u;
    MTU6.TGRD = gu_pulse_u;
    /* End user code. Do not edit comment generated here */
}
```

4.2.8 連続スキャンモード S12AD

以下に、コード生成ツールが連続スキャンモード S12AD 用として出力する API 関数の一覧を示します。

表 4.8 連続スキャンモード S12AD 用 API 関数

API 関数名	機能概要
R_<Config_S12AD0>_Create	連続スキャンモード S12AD を制御するうえで必要となる初期化処理を行います。
R_<Config_S12AD0>_Start	A/D 変換を開始します。
R_<Config_S12AD0>_Stop	A/D 変換を終了します。
R_<Config_S12AD0>_Get_ValueResult	変換結果を獲得します。
R_<Config_S12AD0>_Set_CompareValue	コンペアレベルの設定を行います。
R_<Config_S12AD0>_Set_CompareAValue	ウィンドウ A コンペアレベルの設定を行います。
R_<Config_S12AD0>_Set_CompareBValue	ウィンドウ B コンペアレベルの設定を行います。
R_<Config_S12AD0>_Create_UserInit	連続スキャンモード S12AD に関するユーザ独自の初期化処理を行います。
r_<Config_S12AD0>_interrupt	スキャン終了割り込みの発生に伴う処理を行います。
r_<Config_S12AD0>_compare_interrupt	コンペア割り込みの発生に伴う処理を行います。
r_<Config_S12AD0>_compare_interruptA	ウィンドウ A コンペア割り込みの発生に伴う処理を行います。
r_<Config_S12AD0>_compare_interruptB	ウィンドウ B コンペア割り込みの発生に伴う処理を行います。

R_<Config_S12AD0>_Create

連続スキャンモード S12AD を制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void    R_<Config_S12AD0>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_S12AD0>_Start

A/D 変換を開始します。

[指定形式]

void	R_<Config_S12AD0>_Start (void);

[引数]

なし

[戻り値]

なし

R_<Config_S12AD0>_Stop

A/D 変換を終了します。

[指定形式]

void	R_<Config_S12AD0>_Stop (void);

[引数]

なし

[戻り値]

なし

R_<Config_S12AD0>_Get_ValueResult

変換結果を獲得します。

[指定形式]

```
void    R_<Config_S12AD0>_Get_ValueResult ( ad_channel_t channel, uint16_t * const buffer );
```

[引数]

RX130，RX230/RX231 の場合

I/O	引数	説明
I	ad_channel_t *channel*;	チャネル番号 ADCHANNEL0 ：入力チャネル AN000 ADCHANNEL1 ：入力チャネル AN001 ADCHANNEL2 ：入力チャネル AN002 ADCHANNEL3 ：入力チャネル AN003 ADCHANNEL4 ：入力チャネル AN004 ADCHANNEL5 ：入力チャネル AN005 ADCHANNEL6 ：入力チャネル AN006 ADCHANNEL7 ：入力チャネル AN007 ADCHANNEL16 ：入力チャネル AN016 ADCHANNEL17 ：入力チャネル AN017 ADCHANNEL18 ：入力チャネル AN018 ADCHANNEL19 ：入力チャネル AN019 ADCHANNEL20 ：入力チャネル AN020 ADCHANNEL21 ：入力チャネル AN021 ADCHANNEL22 ：入力チャネル AN022 ADCHANNEL23 ：入力チャネル AN023 ADCHANNEL24 ：入力チャネル AN024 ADCHANNEL25 ：入力チャネル AN025 ADCHANNEL26 ：入力チャネル AN026 ADCHANNEL27 ：入力チャネル AN027 ADCHANNEL28 ：入力チャネル AN028 ADCHANNEL29 ：入力チャネル AN029 ADCHANNEL30 ：入力チャネル AN030 ADCHANNEL31 ：入力チャネル AN031 ADTEMPSENSOR ：拡張アナログ入力（温度センサ出力） ADINTERREFVOLT ：拡張アナログ入力（内部基準電圧） ADSELFDIAGNOSIS ：自己診断結果 ADDATADUPLICATION ：ダブルトリガモード結果
O	uint16_t * const *buffer*;	獲得した変換結果を格納する領域へのポインタ

そのほかのデバイスの場合

I/O	引数	説明
I	ad_channel_t *channel*;	チャネル番号 ADCHANNEL0 : 入力チャネル AN000 ADCHANNEL1 : 入力チャネル AN001 ADCHANNEL2 : 入力チャネル AN002 ADCHANNEL3 : 入力チャネル AN003 ADCHANNEL4 : 入力チャネル AN004 ADCHANNEL5 : 入力チャネル AN005 ADCHANNEL6 : 入力チャネル AN006 ADCHANNEL7 : 入力チャネル AN007 ADCHANNEL8 : 入力チャネル AN008 ADCHANNEL9 : 入力チャネル AN009 ADCHANNEL10 : 入力チャネル AN010 ADCHANNEL11 : 入力チャネル AN011 ADCHANNEL12 : 入力チャネル AN012 ADCHANNEL13 : 入力チャネル AN013 ADCHANNEL14 : 入力チャネル AN014 ADCHANNEL15 : 入力チャネル AN015 ADCHANNEL16 : 入力チャネル AN016 ADCHANNEL17 : 入力チャネル AN017 ADCHANNEL18 : 入力チャネル AN018 ADCHANNEL19 : 入力チャネル AN019 ADCHANNEL20 : 入力チャネル AN020 ADTEMPSENSOR : 拡張アナログ入力（温度センサ出力） ADINTERREFVOLT : 拡張アナログ入力（内部基準電圧） ADSELFDIAGNOSIS : 自己診断結果 ADDATADUPLICATION : ダブルトリガモード結果 ADDATADUPLICATIONA : ダブルトリガモード A 結果 ADDATADUPLICATIONB : ダブルトリガモード B 結果
O	uint16_t * const *buffer*;	獲得した変換結果を格納する領域へのポインタ

[戻り値]

なし

R_<Config_S12AD0>_Set_CompareValue

コンペアレベルの設定を行います。

[指定形式]

```
void      R_<Config_S12AD0>_Set_CompareValue ( uint16_t reg_value0, uint16_t reg_value1);
```

[引数]

I/O	引数	説明
I	uint16_t *reg_value0*;	コンペアレジスタ 0 に設定する値
I	uint16_t *reg_value1*;	コンペアレジスタ 1 に設定する値

[戻り値]

なし

R_<Config_S12AD0>_Set_CompareAValue

ウィンドウ A コンペアレベルの設定を行います。

[指定形式]

void R_<Config_S12AD0>_Set_CompareAValue (uint16_t *reg_value0,* uint16_t *reg_value1*);

[引数]

I/O	引数	説明
I	uint16_t *reg_value0*;	コンペアレジスタ 0 に設定する値
I	uint16_t *reg_value1*;	コンペアレジスタ 1 に設定する値

[戻り値]

なし

R_<Config_S12AD0>_Set_CompareBValue

ウィンドウ B コンペアレベルの設定を行います。

[指定形式]

void R_<Config_S12AD0>_Set_CompareBValue (uint16_t *reg_value0*, uint16_t *reg_value1*);

[引数]

I/O	引数	説明
I	uint16_t *reg_value0*;	コンペアレジスタ 0 に設定する値
I	uint16_t *reg_value1*;	コンペアレジスタ 1 に設定する値

[戻り値]

なし

R_<Config_S12AD0>_Create_UserInit

連続スキャンモード S12AD に関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_S12AD0>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void    R_<Config_S12AD0>_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_S12AD0>_interrupt

スキャン終了割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_S12AD0>_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_S12AD0>_compare_interrupt

コンペア割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_S12AD0>_compare_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_S12AD0>_compare_interruptA

ウィンドウ A コンペア割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_S12AD0>_compare_interruptA ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_S12AD0>_compare_interruptB

ウィンドウ B コンペア割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_S12AD0>_compare_interruptB ( void );
```

[引数]

なし

[戻り値]

なし

使用例

初期設定コンペア条件に一致した AD 変換結果を取得した後、コンペア条件値を変更する

main.c

```
#include "r_smc_entry.h"
void main(void)
{
    /* Start the AD0 converter */
    R_Config_S12AD0_Start();

    while (1U)
    {
        nop();
    }
}
```

Config_S12AD0_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint16_t g_s12ad0_ch000_value;
/* End user code. Do not edit comment generated here */

void r_Config_S12AD0_compare_interrupt(void)
{
    /* Start user code for r_Config_S12AD0_compare_interrupt. Do not edit comment generated here */
    /* Stop the AD0 converter */
    R_Config_S12AD0_Stop();

    /* Get result from the AD0 channel 0 (AN000) converter */
    R_Config_S12AD0_Get_ValueResult(ADCHANNEL0, (uint16_t *)&g_s12ad0_ch000_value);

    /* Set reference data for AD0 comparison */
    R_Config_S12AD0_Set_CompareValue(1000U, 3000U);

    /* Clear the compare flag */
    S12AD.ADCMPSR0.WORD = 0x00U;

    /* Start the AD0 converter */
    R_Config_S12AD0_Start();
    /* End user code. Do not edit comment generated here */
}
```

4.2.9 CRC 演算器

以下に、コード生成ツールが CRC 演算器用として出力する API 関数の一覧を示します。

表 4.9 CRC 演算器用 API 関数

API 関数名	機能概要
R_<Config_CRC>_SetCRC8	8 ビット CRC 演算（多項式：X8 + X2 + X + 1）を実施するうえで必要となる CRC 演算器の初期化処理を行います。
R_<Config_CRC>_SetCRC16	16 ビット CRC 演算（多項式：X16 + X15 + X2 + 1）を実施するうえで必要となる CRC 演算器の初期化処理を行います。
R_<Config_CRC>_SetCCITT	16 ビット CRC 演算（多項式：X16 + X12 + X5 + 1）を実施するうえで必要となる CRC 演算器の初期化処理を行います。
R_<Config_CRC>_SetCRC32	32 ビット CRC 演算（多項式：X32 + X26 + X23 + X22 + X16 + X12 + X11 + X10 + X8 + X7 + X5 + X4 + X2 + X + 1）を実施するうえで必要となる CRC 演算器の初期化処理を行います。
R_<Config_CRC>_SetCRC32C	32 ビット CRC 演算（多項式：X32 + X28 + X27 + X26 + X25 + X23 + X22 + X20 + X19 + X18 + X14 + X13 + X11 + X10 + X9 + X8 + X6 + 1）を実施するうえで必要となる CRC 演算器の初期化処理を行います。
R_<Config_CRC>_Input_Data	CRC 演算を行うデータの初期値を設定します。
R_<Config_CRC>_Get_Result	演算結果を獲得します。

R_<Config_CRC>_SetCRC8

8 ビット CRC 演算（多項式：$X^8 + X^2 + X + 1$）を実施するうえで必要となる CRC 演算器の初期化処理を行います。

[指定形式]

```
void        R_<Config_CRC>_SetCRC8 ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_CRC>_SetCRC16

16 ビット CRC 演算（多項式：$X^{16} + X^{15} + X^2 + 1$）を実施するうえで必要となる CRC 演算器の初期化処理を行います。

[指定形式]

```
void        R_<Config_CRC>_SetCRC16 ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_CRC>_SetCCITT

16 ビット CRC 演算（多項式：$X^{16} + X^{12} + X^5 + 1$）を実施するうえで必要となる CRC 演算器の初期化処理を行います。

[指定形式]

```
void        R_<Config_CRC>_SetCCITT ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_CRC>_SetCRC32

32 ビット CRC 演算（多項式：$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$）を実施するうえで必要となる CRC 演算器の初期化処理を行います。

[指定形式]

```
void        R_<Config_CRC>_SetCRC32 ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_CRC>_SetCRC32C

32 ビット CRC 演算（多項式 : $X^{32} + X^{28} + X^{27} + X^{26} + X^{25} + X^{23} + X^{22} + X^{20} + X^{19} + X^{18} + X^{14} + X^{13} + X^{11} + X^{10} + X^{9} + X^{8} + X^{6} + 1$）を実施するうえで必要となる CRC 演算器の初期化処理を行います。

[指定形式]

```
void        R_<Config_CRC>_SetCRC32C ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_CRC>_Input_Data

CRC 演算を行うデータの初期値を設定します。

[指定形式]

void	R_<Config_CRC>_Input_Data (uint8_t *data*);

void	R_<Config_CRC>_Input_Data (uint32_t *data*);

[引数]

I/O	引数	説明
I	uint8_t *data*;	CRC 演算を行うデータの初期値

I/O	引数	説明
I	uint32_t *data*;	CRC 演算を行うデータの初期値

備考 デバイスグループにより、引数のサイズが異なります。

[戻り値]

なし

R_<Config_CRC>_Get_Result

演算結果を獲得します。

[指定形式]

void	R_<Config_CRC>_Get_Result (uint16_t * const *result*);

void	R_<Config_CRC>_Get_Result (uint32_t * const *result*);

[引数]

I/O	引数	説明
O	uint16_t * const *result*;	獲得した演算結果を格納する領域へのポインタ

I/O	引数	説明
O	uint32_t * const *result*;	獲得した演算結果を格納する領域へのポインタ

備考 デバイスグループにより、引数のサイズが異なります。

[戻り値]

なし

使用例 1

CRC コードを生成し、送信データに追加する

tx_func.c

```
#include "r_smc_entry.h"
volatile uint8_t tx_buf[2];
volatile uint16_t result;
void tx_func(void)
{
    /* Set CRC module using CRC8 algorithm */
    R_Config_CRC_SetCRC8();

    /* Restore transmit data */
    tx_buf[0] = 0xF0;

    /* Write data to CRC input register */
    R_Config_CRC_Input_Data(tx_buf[0]);

    /* Get result from CRC output register */
    R_Config_CRC_Get_Result((uint16_t *)&result);

    /* Restore CRC code */
    tx_buf[1] = (uint8_t)(result);

    /*** Transmit "tx_buf" ***/
}
```

使用例 2

解析した受信データから CRC コードを生成し、受信データが正しいかチェックする

rx_func.c

```
#include "r_smc_entry.h"
volatile uint8_t rx_buf[2];
volatile uint16_t result;
volatile uint8_t err_f;
void rx_func(void)
{
    /* Clare error flag */
    err_f = 0U;

    /*** Receive (Restore the receive data in "rx_buf") ***/

    /* Set CRC module using CRC8 algorithm */
    R_Config_CRC_SetCRC8();

    /* Write data to CRC input register */
    R_Config_CRC_Input_Data(rx_buf[0]);

    /* Get result from CRC output register */
    R_Config_CRC_Get_Result((uint16_t *)&result);

    /* Check the receive data */
    if (rx_buf[1] != (uint8_t)(result))
    {
        /* Set error flag */
        err_f = 1U;
    }
}
```

4.2.10 D/A コンバータ

以下に、コード生成ツールが D/A コンバータ用として出力する API 関数の一覧を示します。

表 4.10 D/A コンバータ用 API 関数

API 関数名	機能概要
R_<Config_DA>_Create	D/A コンバータを制御するうえで必要となる初期化処理を行います。
R_<Config_DA>n_Start	D/A 変換を開始します。
R_<Config_DA>n_Stop	D/A 変換を終了します。
R_<Config_DA>n_Set_ConversionValue	D/A 変換を行うデータを設定します。
R_<Config_DA>_Sync_Start	D/A 一括変換を開始します。
R_<Config_DA>_Sync_Stop	D/A 一括変換を終了します。
R_<Config_DA>_Create_UserInit	D/A コンバータに関するユーザ独自の初期化処理を行います。

R_<Config_DA>_Create

D/A コンバータを制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_<Config_DA>*n*_Start

D/A 変換を開始します。

[指定形式]

void	R_<Config_DA>*n*_Start (void);

備考 *n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_<Config_DA>*n*_Stop

D/A 変換を終了します。

[指定形式]

void	R_<Config_DA>*n*_Stop (void);

備考 *n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_<Config_DA>*n*_Set_ConversionValue

D/A 変換を行うデータを設定します。

[指定形式]

void	R_<Config_DA>*n*_Set_ConversionValue (uint16_t *reg_value*);

備考 *n* はチャネル番号を意味します。

[引数]

I/O	引数	説明
I	uint16_t *reg_value*;	D/A 変換を行うデータ

[戻り値]

なし

R_<Config_DA>_Sync_Start

D/A 一括変換を開始します。

[指定形式]

void	R_<Config_DA>_Sync_Start (void);

[引数]

なし

[戻り値]

なし

R_<Config_DA>_Sync_Stop

D/A 一括変換を終了します。

[指定形式]

void	R_<Config_DA>_Sync_Stop (void);

[引数]

なし

[戻り値]

なし

R_<Config_DA>_Create_UserInit

D/A コンバータに関するユーザ独自の初期化処理を行います。

備考 本 API 関数は、R_<Config_DA>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_DA>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

使用例

チャネル 0，1 の D/A 変換を一括許可する

main.c

```
#include "r_smc_entry.h"
void main(void)
{
    /* Set the DA0 converter value */
    R_<Config_DA>0_Set_ConversionValue(1000U);

    /* Set the DA1 converter value */
    R_<Config_DA>1_Set_ConversionValue(2000U);

    /* Enable the DA0, DA1 synchronize converter */
    R_<Config_DA>_Sync_Start();

    while (1U)
    {
        nop();
    }
}
```

4.2.11 データ演算回路

以下に、コード生成ツールがデータ演算回路用として出力する API 関数の一覧を示します。

表 4.11 データ演算回路用 API 関数

API 関数名	機能概要
R_<Config_DOC>_Create	データ演算回路を制御するうえで必要となる初期化処理を行います。
R_<Config_DOC>SetMode	動作モード、およびデータ演算を行うデータの初期値を設定します。
R_<Config_DOC>_WriteData	データ演算を行う値（比較／加算／減算する値）を設定します。
R_<Config_DOC>_GetResult	演算結果を獲得します。
R_<Config_DOC>_ClearFlag	データ演算回路フラグをクリアします。
R_<Config_DOC>_Create_UserInit	データ演算回路に関するユーザ独自の初期化処理を行います。
r_<Config_DOC>_dopcf_interrupt	データ演算回路割り込みの発生に伴う処理を行います。 （デバイスグループにより、API 関数名が異なります。）
r_<Config_DOC>_dopci_interrupt	

R_<Config_DOC>_Create

データ演算回路を制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_<Config_DOC>_SetMode

動作モード、およびデータ演算を行うデータの初期値を設定します。

本 API 関数はデバイスの DOC の機能に応じて、2 つのパラメータで定義されている API 関数と、3 つのパラメータで定義されている API 関数があります。

- 2 つのパラメータ（ “mode” と “value” ）で定義されている R_<Config_DOC>_SetMode

備考 1. 動作モード *mode* にデータ比較モード COMPARE_MISMATCH、または COMPARE_MATCH が指定された場合、基準となる 16 ビットのデータ *value* が DOC データセッティングレジスタ（DODSR）に格納されます。

備考 2. 動作モード *mode* にデータ加算モード ADDITION、またはデータ減算モード SUBTRACTION が指定された場合、初期値として 16 ビットのデータ *value* が DOC データセッティングレジスタ（DODSR）に格納されます。

[指定形式]

void	R_<Config_DOC>_SetMode (doc_mode_t *mode*, uint16_t *value*);

[引数]

I/O	引数	説明
I	doc_mode_t *mode*;	動作モード（検出条件を含む）の種類 COMPARE_MISMATCH ：データ比較モード（不一致） COMPARE_MATCH ：データ比較モード（一致） ADDITION ：データ加算モード SUBTRACTION ：データ減算モード
I	uint16_t *value*;	比較演算の基準値、加算・減算の演算結果

[戻り値]

なし

- 3 つのパラメータ（ “mode” 、“value1” と “value2” ）で定義されている R_<Config_DOC>_SetMode

備考 1. 動作モード *mode* にデータ比較モードが指定されて、値が COMPARE_NEQ(等しくない) または COMPARE_EQ（等しい）または COMPARE_GT（より大きい）または COMPARE_LT（より小さい）の場合、16 ビット/32 ビットのデータ *value* が DOC データセッティングレジスタ 0（DODSR0）に格納されます。

備考 2. 動作モード *mode* にデータ加算モード ADDITION、またはデータ減算モード SUBTRACTION が指定された場合、初期値として 16 ビット/32 ビットのデータ *value* が DOC データセッティングレジスタ 0（DODSR0）に格納されます。

備考 3. 動作モード *mode* にデータ比較モード COMPARE_IN_RANGE、または COMPARE_OUT_RANGE が指定された場合、16 ビット/32 ビットのデータ *value1*(範囲の下限値) が DOC データセッティングレジスタ 0（DODSR0）に格納されます。16 ビット/32 ビットのデータ *value2*(範囲の上限値) が DOC データセッティングレジスタ 1（DODSR1）に格納されます。

[指定形式]

void	R_<Config_DOC>_SetMode (doc_mode_t *mode*, type *value1,* type *value2*);

[引数]

I/O	引数	説明
I	doc_mode_t *mode*;	動作モード（検出条件を含む）の種類 COMPARE_NEQ :データ比較モード(等しくない) COMPARE_EQ : データ比較モード（等しい） COMPARE_GT :データ比較モード(より大きい) COMPARE_LT :データ比較モード(より小さい) COMPARE_IN_RANGE : データ比較モード（範囲内） COMPARE_OUT_RANGE : データ比較モード（範囲外） ADDITION : データ加算モード SUBTRACTION : データ減算モード
I	type *value1*;	- 範囲比較を除く比較演算の基準値、加算・減算の演算結果 - 範囲比較の下限値 - “type” は “uint16_t” もしくは “uint32_t” に指定できる
I	type *value2*;	- 範囲比較の上限値 - “type” は “uint16_t” もしくは “uint32_t” に指定できる

[戻り値]

なし

R_<Config_DOC>_WriteData

データ演算を行う値（比較／加算／減算する値）を設定します。

[指定形式]

void	R_<Config_DOC>_WriteData (type *data*);

[引数]

I/O	引数	説明
I	type *data*;	データ演算を行う値 “type” は “uint16_t” もしくは “uint32_t” に指定できる

[戻り値]

なし

R_<Config_DOC>_GetResult

演算結果を獲得します。

[指定形式]

void R_<Config_DOC>_GetResult (type * const *data*);

[引数]

I/O	引数	説明
O	type * const *data*;	獲得した演算結果を格納する領域へのポインタ “type” は “uint16_t” もしくは “uint32_t” に指定できる

[戻り値]

なし

R_<Config_DOC>_ClearFlag

データ演算回路フラグをクリアします。

[指定形式]

void	R_<Config_DOC>_ClearFlag (void);

[引数]

なし

[戻り値]

なし

R_<Config_DOC>_Create_UserInit

データ演算回路に関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_DOC>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_DOC>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

r_<Config_DOC>_dopcf_interrupt

r_<Config_DOC>_dopci_interrupt

データ演算回路割り込みの発生に伴う処理を行います。

備考 本 API 関数は、データの比較結果が検出条件を満足した場合、データの加算結果が 0xFFFF よりも大きくなった場合、またはデータの減算結果が 0x0 よりも小さくなった場合に発生するデータ演算回路割り込みに対応した割り込み処理として呼び出されます。

[指定形式]

void	r_<Config_DOC>_dopcf_interrupt (void);

void	r_<Config_DOC>_dopci_interrupt (void);

備考 デバイスグループにより、API 関数名が異なります。

[引数]

なし

[戻り値]

なし

使用例

データ加算モードで配列データを加算し、“FFFFh”より大きくなったとき割り込みで加算結果を取得する
その後データ比較不一致モードに変更し、配列データに"000h"以外を検出した場合に割り込みを発生させる

main.c

```
#include "r_smc_entry.h"
extern volatile uint16_t data[16];
void main(void)
{
    uint8_t cnt;
    while (1U)
    {
        for (cnt = 0; cnt < 16U; cnt++)
        {
            /* Write new data to compare */
            R_<Config_DOC>_WriteData(data[cnt]);
        }
    }
}
```

Config_DOC_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint16_t data[16];
volatile uint16_t result;
/* End user code. Do not edit comment generated here */

void r_<Config_DOC>_dopci_interrupt(void)
{
    /* Start user code for r_<Config_DOC_dopci_interrupt. Do not edit comment generated here */
    /* Get result */
    R_<Config_DOC>_GetResult((uint16_t *)&result);

    /* Configure the operation mode of DOC */
    R_<Config_DOC>_SetMode(COMPARE_MISMATCH, 0x0000);

    /* Clear DOPCI flag */
    R_<Config_DOC>_ClearFlag();
    /* End user code. Do not edit comment generated here */
}
```

4.2.12 データトランスファコントローラ

以下に、コード生成ツールがデータトランスファコントローラ用として出力する API 関数の一覧を示します。

表 4.12 データトランスファコントローラ用 API 関数

API 関数名	機能概要
R_<Config_DTC>_Create	データトランスファコントローラを制御するうえで必要となる初期化処理を行います。
R_<Config_DTC>_Start	データトランスファコントローラの起動を許可します。
R_<Config_DTC>_Stop	データトランスファコントローラの起動を禁止します。
R_<Config_DTC>_Create_UserInit	データトランスファコントローラに関するユーザ独自の初期化処理を行います。

R_<Config_DTC>_Create

データトランスファコントローラを制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_<Config_DTC>_Start

データトランスファコントローラの起動を許可します。

備考　本 API 関数では、選択した起動要因に対応する DTCE ビットを操作することにより、データトランスファコントローラの起動許可を実現しています。

[指定形式]

void	R_<Config_DTC>_Start (void);

[引数]

なし

[戻り値]

なし

R_<Config_DTC>_Stop

データトランスファコントローラの起動を禁止します。

備考　本 API 関数では、選択した起動要因に対応する DTCE ビットを操作することにより、データトランスファコントローラの起動禁止を実現しています。

[指定形式]

void	R_<Config_DTC>_Stop (void);

[引数]

なし

[戻り値]

なし

R_<Config_DTC>_Create_UserInit

データトランスファコントローラに関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_DTC>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_DTC>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

使用例

コンペアマッチ割り込みで DTC データ転送を開始する

main.c

```
#include "r_smc_entry.h"
void main(void)
{
    /* Start CMT channel 0 counter */
    R_Config_CMT0_Start();

    /* Enable operation of transfer data DTC */
    R_Config_DTC_Start();

    while (1U)
    {
        nop();
    }
}
```

4.2.13 デッドタイム補償用カウンタ

以下に、コード生成ツールがデッドタイム補償用カウンタ用として出力する API 関数の一覧を示します。

表 4.13 デッドタイム補償用カウンタ用 API 関数

API 関数名	機能概要
R_<Config_MTU5>_Create	デッドタイム補償用カウンタを制御するうえで必要となる初期化処理を行います。
R_<Config_MTU5>_U5_Start	カウンタにより U 相カウントを開始します。
R_<Config_MTU5>_U5_Stop	カウンタにより U 相カウントを終了します。
R_<Config_MTU5>_V5_Start	カウンタにより V 相カウントを開始します。
R_<Config_MTU5>_V5_Stop	カウンタにより V 相カウントを終了します。
R_<Config_MTU5>_W5_Start	カウンタにより W 相カウントを開始します。
R_<Config_MTU5>_W5_Stop	カウンタにより W 相カウントを終了します。
R_<Config_MTU5>_Create_UserInit	デッドタイム補償用カウンタに関するユーザ独自の初期化処理を行います。
r_<Config_MTU5>_tgimn_interrupt	インプットキャプチャ割り込みの発生に伴う処理を行います。

R_<Config_MTU5>_Create

デッドタイム補償用カウンタを制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void    R_<Config_MTU5>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_MTU5>_U5_Start

U 相カウントを開始します。

[指定形式]

```
void        R_<Config_MTU5>_U5_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_MTU5>_U5_Stop

U 相カウントを終了します。

[指定形式]

void	R_<Config_MTU5>_U5_Stop (void);

[引数]

なし

[戻り値]

なし

R_<Config_MTU5>_V5_Start

V 相カウントを開始します。

[指定形式]

```
void        R_<Config_MTU5>_V5_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_MTU5>_V5_Stop

V 相カウントを終了します。

[指定形式]

void	R_<Config_MTU5>_V5_Stop (void);

[引数]

なし

[戻り値]

なし

R_<Config_MTU5>_W5_Start

W 相カウントを開始します。

[指定形式]

```
void        R_<Config_MTU5>_W5_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_MTU5>_W5_Stop

W 相カウントを終了します。

[指定形式]

```
void        R_<Config_MTU5>_W5_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_MTU5>_Create_UserInit

デッドタイム補償用カウンタに関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_MTU5>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_MTU5>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

r_<Config_MTU5>_tgi*mn*_interrupt

インプットキャプチャ割り込みの発生に伴う処理を行います。

備考　本 API 関数は、入力信号のエッジを検出した場合に発生するインプットキャプチャ割り込みに対応した割り込み処理として呼び出されます。

[指定形式]

void	r_<Config_MTU5>_tgi*mn*_interrupt (void);

備考　*n* はチャネル番号を、*m* はタイマジェネラルレジスタ番号を意味します。

[引数]

なし

[戻り値]

なし

使用例

MTU6，MTU7 を使用した相補 PWM 動作時の PWM 出力波形に対するデッドタイムを補償する

main.c

```
#include "r_smc_entry.h"
void main(void)
{
    /* Start the MTU5 channel counter */
    R_Config_MTU5_Start();

    /* Start the MTU6 channel counter */
    R_Config_MTU6_MTU7_Start();

    while (1U)
    {
        nop();
    }
}
```

Config_MTU5_user.c

```
static void r_<Config_MTU5_tgiu5_interrupt(void)
{
    /* Start user code for r_<Config_MTU5_tgiu5_interrupt. Do not edit comment generated here */
    /* Write the corrected value */
    if ( MTU6.TGRB > MTU5.TGRU )
    {
        MTU6.TGRD = (MTU6.TGRB - MTU5.TGRU);
    }
    /* End user code. Do not edit comment generated here */
}
```

4.2.14 DMA コントローラ

以下に、コード生成ツールが DMA コントローラ用として出力する API 関数の一覧を示します。

表 4.14 DMA コントローラ用 API 関数

API 関数名	機能概要
R_<Config_DMAC0>_Create	DMA コントローラを制御するうえで必要となる初期化処理を行います。
R_<Config_DMAC0>_Start	DMA 起動を開始します。
R_<Config_DMAC0>_Stop	DMA 起動を終了します。
R_<Config_DMAC0>_Set_SoftwareTrigger	ソフトウェア転送要求をセットします。
R_<Config_DMAC0>_Clear_SoftwareTrigger	ソフトウェア転送要求をクリアします。
R_<Config_DMAC0>_Create_UserInit	DMA コントローラに関するユーザ独自の初期化処理を行います。
r_<Config_DMAC0>_dmacni_interrupt	チャネル *n* の転送終了割り込みに伴う処理を行います。(*n* = 0 ~ 3)
r_dmacn_callback_transfer_end	チャネル *n* の転送終了割り込みに伴う処理を行います。(*n* = 0 ~ 3)
r_dmacn_callback_transfer_escape_end	チャネル *n* のエスケープ転送終了割り込みに伴う処理を行います。(*n* = 0 ~ 3)
r_dmac_dmac74i_interrupt	チャネル 4 ~ 7 の転送終了割り込みに伴う処理を行います。
r_dmacn_callback_transfer_end	チャネル *n* の転送終了割り込みに伴う処理を行います。(*n* = 4 ~ 7)
r_dmacn_callback_transfer_escape_end	チャネル *n* のエスケープ転送終了割り込みに伴う処理を行います。(*n* = 4 ~ 7)

R_<Config_DMAC0>_Create

DMA コントローラを制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void    R_<Config_DMAC0>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_DMAC0>_Start

DMA 起動を開始します。

[指定形式]

```
void        R_<Config_DMAC0>_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_DMAC0>_Stop

DMA 起動を終了します。

[指定形式]

```
void        R_<Config_DMAC0>_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_DMAC0>_Set_SoftwareTrigger

ソフトウェア転送要求をセットします。

備考 ソフトウェアトリガを使用して連続で転送を行う場合、CLRS ビットに直接 1 を設定してください。その後、この関数を使用することで転送完了後に連続して同じ転送が行われます。R_<Config_DMAC0>_Clear_SoftwareTrigger を使用することで転送を停止することができます。

[指定形式]

void	R_<Config_DMAC0>_Set_SoftwareTrigger (void);

[引数]

なし

[戻り値]

なし

R_<Config_DMAC0>_Clear_SoftwareTrigger

ソフトウェア転送要求をクリアします。

[指定形式]

void	R_<Config_DMAC0>_Clear_SoftwareTrigger (void);

[引数]

なし

[戻り値]

なし

R_<Config_DMAC0>_Create_UserInit

DMA コントローラに関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_DMAC0>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_DMAC0>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

r_<Config_DMAC0>_dmac*n*i_interrupt

チャネル *n* の転送終了割り込みに伴う処理を行います。

[指定形式]

void	r_<Config_DMAC0>_dmac*n*i_interrupt (void);

備考 *n* はチャネル番号を意味します。(*n* = 0 ~ 3)

[引数]

なし

[戻り値]

なし

r_dmac*n*_callback_transfer_end

チャネル *n* の転送終了割り込みに伴う処理を行います。

備考 API 関数は、チャネル *n* の転送終了割り込みに対応した割り込み処理 r_<Config_DMAC0>_dmacni_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	r_dmac*n*_callback_transfer_end (void);

備考 *n* はチャネル番号を意味します。(*n* = 0 ~ 3)

[引数]

なし

[戻り値]

なし

r_dmac*n*_callback_transfer_escape_end

チャネル *n* のエスケープ転送終了割り込みに伴う処理を行います。

備考 API 関数は、チャネル *n* のエスケープ転送終了割り込みに対応した割り込み処理 r_<Config_DMAC0>_dmacni_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	r_dmac*n*_callback_transfer_escape_end (void);

備考 *n* はチャネル番号を意味します。(*n* = 0 ~ 3)

[引数]

なし

[戻り値]

なし

r_dmac_dmac74i_interrupt

チャネル 4～7 の転送終了割り込みに伴う処理を行います。

[指定形式]

```
void    r_dmac_dmac74i_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_dmac*n*_callback_transfer_end

チャネル *n* の転送終了割り込みに伴う処理を行います。

備考 API 関数は、チャネル *n* の転送終了割り込みに対応した割り込み処理 r_dmac_dmac74i_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	r_dmac*n*_callback_transfer_end (void);

備考 *n* はチャネル番号を意味します。(*n* = 4 ~ 7)

[引数]

なし

[戻り値]

なし

r_dmac*n*_callback_transfer_escape_end

チャネル *n* のエスケープ転送終了割り込みに伴う処理を行います。

備考 API 関数は、チャネル *n* のエスケープ転送終了割り込みに対応した割り込み処理 r_dmac_dmac74i_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	r_dmac*n*_callback_transfer_escape_end (void);

備考 *n* はチャネル番号を意味します。(*n* = 4 ~ 7)

[引数]

なし

[戻り値]

なし

使用例

コンペアマッチ割り込みで転送開始し、転送完了でフラグを立てる

main.c

```
#include "r_smc_entry.h"
void main(void)
{
    /* Start CMT channel 0 counter */
    R_Config_CMT0_Start();

    /* Enable the DMAC0 activation */
    R_Config_DMAC0_Start();

    while (1U)
    {
        nop();
    }
}
```

Config_DMAC0_user.c

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

void R_Config_DMAC0_Create_UserInit(void)
{
    /* Start user code for user init. Do not edit comment generated here */
    /* Clear the flag */
    g_dmac0_f = 0U;
    /* End user code. Do not edit comment generated here */
}

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

Config_DMAC0.h

```
#define _DMAC0_ACTIVATION_SOURCE      (28U) /* Please assign dynamic vector in interrupt tab */
```

※この記述は保護されません。再度コード生成した場合は再度記述してください。

4.2.15 イベントリンクコントローラ

以下に、コード生成ツールがイベントリンクコントローラ用として出力する API 関数の一覧を示します。

表 4.15 イベントリンクコントローラ用 API 関数

API 関数名	機能概要
R_<Config_ELC>_Create	イベントリンクコントローラ（ELC）を制御するうえで必要となる初期化処理を行います。
R_<Config_ELC>_Start	周辺機能間の連携を開始します。
R_<Config_ELC>_Stop	周辺機能間の連携を終了します。
R_<Config_ELC>_GenerateSoftwareEvent	ソフトウェアイベントを発生させます。
R_<Config_ELC>_Set_PortBuffern	ポートバッファに値を書き込みます。
R_<Config_ELC>_Get_PortBuffern	ポートバッファの値を読み出します。
R_<Config_ELC>_Create_UserInit	イベントリンクコントローラ（ELC）に関するユーザ独自の初期化処理を行います。
r_<Config_ELC>_elsrni_interrupt	イベント割り込みの発生に伴う処理を行います。

R_<Config_ELC>_Create

イベントリンクコントローラ（ELC）を制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_<Config_ELC>_Start

周辺機能間の連携を開始します。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_<Config_ELC>_Stop

周辺機能間の連携を終了します。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_<Config_ELC>_GenerateSoftwareEvent

ソフトウェアイベントを発生させます。

[指定形式]

void	R_<Config_ELC>_GenerateSoftwareEvent (void);

[引数]

なし

[戻り値]

なし

R_<Config_ELC>_Set_PortBuffer*n*

ポートバッファに値を書き込みます。

[指定形式]

void	R_<Config_ELC>_Set_PortBuffer*n* (uint8_t *value*);

備考 *n* はポート番号を意味します。

[引数]

I/O	引数	説明
I	uint8_t *value*;	書き込む値

[戻り値]

なし

R_<Config_ELC>_Get_PortBuffer*n*

ポートバッファの値を読み出します。

[指定形式]

void	R_<Config_ELC>_Get_PortBuffer*n* (uint8_t * const *value*);

備考 *n* はポート番号を意味します。

[引数]

I/O	引数	説明
O	uint8_t * const *value*;	読み出した値を格納する領域へのポインタ

[戻り値]

なし

R_<Config_ELC>_Create_UserInit

イベントリンクコントローラ（ELC）に関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_ELC>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

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

[引数]

なし

[戻り値]

なし

r_<Config_ELC>_elsr*n*i_interrupt

イベント割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_ELC>_elsr*n*i_interrupt (void);

備考 *n* はイベントリンク設定レジスタ番号を意味します。

[引数]

なし

[戻り値]

なし

使用例

ソフトウェアイベントを発生させ、リンクさせたイベントを発生させる。
次々にリンクさせ、イベント割り込みまで来ると終了。

main.c

```
#include "r_smc_entry.h"
void main(void)
{
    /* Enable all ELC event links */
    R_Config_ELC_Start();

    /* Trigger a software event */
    R_Config_ELC_GenerateSoftwareEvent();

    while (1U)
    {
        nop();
    }
}
```

Config_ELC_user.c

```
static void r_Config_ELC_elsr18i_interrupt(void)
{
    /* Start user code for r_Config_ELC_elsr18i_interrupt. Do not edit comment generated here */
    /* Disable all ELC event links */
    R_Config_ELC_Stop();
    /* End user code. Do not edit comment generated here */
}
```

4.2.16 汎用 PWM タイマ

以下に、コード生成ツールが汎用 PWM タイマ用として出力する API 関数の一覧を示します。

表 4.16 汎用 PWM タイマ用 API 関数

API 関数名	機能概要
R_<Config_GPT0>_Create	汎用 PWM タイマを制御するうえで必要となる初期化処理を行います。
R_<Config_GPT0>_Start	カウントを開始します。
R_<Config_GPT0>_Stop	カウントを終了します。
R_<Config_GPT0>_HardwareStart	割り込みを許可します。
R_<Config_GPT0>_HardwareStop	割り込みを禁止します。
R_<Config_GPT0>_ETGI_Start	外部トリガ立ち下がり／立ち上がり割り込みを許可します。
R_<Config_GPT0>_ETGI_Stop	外部トリガ立ち下がり／立ち上がり割り込みを禁止します。
R_<Config_GPT0>_Software_Clear	カウンタをクリアします。
R_<Config_GPT0>_Create_UserInit	汎用 PWM タイマに関するユーザ独自の初期化処理を行います。
r_<Config_GPT0>_gtcimn_interrupt	インプットキャプチャ／コンペアマッチ割り込みの発生に伴う処理を行います。
r_<Config_GPT0>_gtcivn_interrupt	オーバフロー割り込みの発生に伴う処理を行います。
r_<Config_GPT0>_gtciun_interrupt	アンダフロー割り込みの発生に伴う処理を行います。
r_<Config_GPT0>_gdten_interrupt	デッドタイムエラー割り込みの発生に伴う処理を行います。
r_gpt_etgin_interrupt	外部トリガ立ち下がり割り込みの発生に伴う処理を行います。
r_gpt_etgip_interrupt	外部トリガ立ち上がり割り込みの発生に伴う処理を行います。

R_<Config_GPT0>_Create

汎用 PWM タイマを制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void    R_<Config_GPT0>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_GPT0>_Start

カウントを開始します。

備考　RX26T/RX66T/RX66N/RX72N/RX72M/RX72T を使用する場合、カウント開始要因にソフトウェア要因カウントスタートが設定されておらず、すべての GPT 割り込みが禁止に設定されていると、この関数は空の状態で生成されます。

[指定形式]

```
void    R_<Config_GPT0>_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_GPT0>_Stop

カウントを終了します。

備考　RX26T/RX66T/RX66N/RX72N/RX72M/RX72T を使用する場合、カウント停止要因にソフトウェア要因カウントストップが設定されておらず、すべての GPT 割り込みが禁止に設定されていると、この関数は空の状態で生成されます。

[指定形式]

```
void    R_<Config_GPT0>_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_GPT0>_HardwareStart

割り込みを許可します。

備考　本 API 関数は外部/内部トリガ(ハードウェア要因)によるカウント動作時に割り込みの検出を有効化するために使用します。

[指定形式]

void	R_<Config_GPT0>_HardwareStart (void);

[引数]

なし

[戻り値]

なし

R_<Config_GPT0>_HardwareStop

割り込みを禁止します。

備考　本 API 関数は外部/内部トリガ(ハードウェア要因)によるカウント動作時に割り込みの検出を有効化するために使用します。

[指定形式]

void	R_<Config_GPT0>_HardwareStop (void);

[引数]

なし

[戻り値]

なし

R_<Config_GPT0>_ETGI_Start

外部トリガ立ち下がり／立ち上がり割り込みを許可します。

[指定形式]

void	R_<Config_GPT0>_ETGI_Start (void);

[引数]

なし

[戻り値]

なし

R_<Config_GPT0>_ETGI_Stop

外部トリガ立ち下がり／立ち上がり割り込みを禁止します。

[指定形式]

void	R_<Config_GPT0>_ETGI_Stop (void);

[引数]

なし

[戻り値]

なし

R_<Config_GPT0>_Software_Clear

カウンタをクリアします。

[指定形式]

void	R_<Config_GPT0>_Software_Clear (void);

[引数]

なし

[戻り値]

なし

R_<Config_GPT0>_Create_UserInit

汎用 PWM タイマに関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_GPT0>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_GPT0>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

r_<Config_GPT0>_gtci*mn*_interrupt

インプットキャプチャ／コンペアマッチ割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_GPT0>_gtci*mn*_interrupt (void);

備考　*n* はチャネル番号を、*m* はタイマジェネラルレジスタ番号を意味します。

[引数]

なし

[戻り値]

なし

r_<Config_GPT0>_gtciv*n*_interrupt

オーバフロー割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_GPT0>_gtciv*n*_interrupt (void);

備考 *n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

r_<Config_GPT0>_gtciu*n*_interrupt

アンダフロー割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_GPT0>_gtciu*n*_interrupt (void);

備考 *n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

r_<Config_GPT0>_gdte*n*_interrupt

デッドタイムエラー割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_GPT0>_gdte*n*_interrupt (void);

備考 *n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

r_gpt_etgin_interrupt

外部トリガ立ち下がり割り込みの発生に伴う処理を行います。

[指定形式]

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

[引数]

なし

[戻り値]

なし

r_gpt_etgip_interrupt

外部トリガ立ち上がり割り込みの発生に伴う処理を行います。

[指定形式]

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

[引数]

なし

[戻り値]

なし

使用例

インプットキャプチャ値を取得する

main.c

```
#include "r_smc_entry.h"
void main(void)
{
    /* Start GPT channel 0 counter */
    R_Config_GPT0_Start();

    while (1U)
    {
        nop();
    }
}
```

Config_GPT0_user.c

```
* Start user code for global. Do not edit comment generated here */
volatile uint16_t g_gpt0_capture_value;
/* End user code. Do not edit comment generated here */

static void r_Config_GPT0_gtcia0_interrupt(void)
{
    /* Start user code for r_Config_GPT0_gtcia0_interrupt. Do not edit comment generated here */
    g_gpt0_capture_value = GPT0.GTCCRA;
    /* End user code. Do not edit comment generated here */
}
```

4.2.17 グループスキャンモード S12AD

以下に、コード生成ツールがグループスキャンモード S12AD 用として出力する API 関数の一覧を示します。

表 4.17 グループスキャンモード S12AD 用 API 関数

API 関数名	機能概要
R_<Config_S12AD0>_Create	グループスキャンモード S12AD を制御するうえで必要となる初期化処理を行います。
R_<Config_S12AD0>_Start	A/D 変換を開始します。
R_<Config_S12AD0>_Stop	A/D 変換を終了します。
R_<Config_S12AD0>_Get_ValueResult	変換結果を獲得します。
R_<Config_S12AD0>_Set_CompareValue	コンペアレベルの設定を行います。
R_<Config_S12AD0>_Set_CompareAValue	ウィンドウ A コンペアレベルの設定を行います。
R_<Config_S12AD0>_Set_CompareBValue	ウィンドウ B コンペアレベルの設定を行います。
R_<Config_S12AD0>_Create_UserInit	グループスキャンモード S12AD に関するユーザ独自の初期化処理を行います。
r_<Config_S12AD0>_interrupt	スキャン終了割り込みの発生に伴う処理を行います。
r_<Config_S12AD0>_compare_interrupt	コンペア割り込みの発生に伴う処理を行います。
r_<Config_S12AD0>_compare_interruptA	ウィンドウ A コンペア割り込みの発生に伴う処理を行います。
r_<Config_S12AD0>_compare_interruptB	ウィンドウ B コンペア割り込みの発生に伴う処理を行います。
r_<Config_S12AD0>_groupb_interrupt	グループ B スキャン終了割り込みの発生に伴う処理を行います。
r_<Config_S12AD0>_groupc_interrupt	グループ C スキャン終了割り込みの発生に伴う処理を行います。

R_<Config_S12AD0>_Create

グループスキャンモード S12AD を制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void    R_<Config_S12AD0>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_S12AD0>_Start

A/D 変換を開始します。

[指定形式]

void	R_<Config_S12AD0>_Start (void);

[引数]

なし

[戻り値]

なし

R_<Config_S12AD0>_Stop

A/D 変換を終了します。

[指定形式]

void	R_<Config_S12AD0>_Stop (void);

[引数]

なし

[戻り値]

なし

R_<Config_S12AD0>_Get_ValueResult

変換結果を獲得します。

[指定形式]

```
void    R_<Config_S12AD0>_Get_ValueResult ( ad_channel_t channel, uint16_t * const buffer );
```

[引数]

RX130，RX230/RX231 の場合

I/O	引数	説明
I	ad_channel_t *channel*;	チャネル番号 ADCHANNEL0 ：入力チャネル AN000 ADCHANNEL1 ：入力チャネル AN001 ADCHANNEL2 ：入力チャネル AN002 ADCHANNEL3 ：入力チャネル AN003 ADCHANNEL4 ：入力チャネル AN004 ADCHANNEL5 ：入力チャネル AN005 ADCHANNEL6 ：入力チャネル AN006 ADCHANNEL7 ：入力チャネル AN007 ADCHANNEL16 ：入力チャネル AN016 ADCHANNEL17 ：入力チャネル AN017 ADCHANNEL18 ：入力チャネル AN018 ADCHANNEL19 ：入力チャネル AN019 ADCHANNEL20 ：入力チャネル AN020 ADCHANNEL21 ：入力チャネル AN021 ADCHANNEL22 ：入力チャネル AN022 ADCHANNEL23 ：入力チャネル AN023 ADCHANNEL24 ：入力チャネル AN024 ADCHANNEL25 ：入力チャネル AN025 ADCHANNEL26 ：入力チャネル AN026 ADCHANNEL27 ：入力チャネル AN027 ADCHANNEL28 ：入力チャネル AN028 ADCHANNEL29 ：入力チャネル AN029 ADCHANNEL30 ：入力チャネル AN030 ADCHANNEL31 ：入力チャネル AN031 ADTEMPSENSOR ：拡張アナログ入力（温度センサ出力） ADINTERREFVOLT ：拡張アナログ入力（内部基準電圧） ADSELFDIAGNOSIS ：自己診断結果 ADDATADUPLICATION ：ダブルトリガモード結果
O	uint16_t * const *buffer*;	獲得した変換結果を格納する領域へのポインタ

そのほかのデバイスの場合

I/O	引数	説明
I	ad_channel_t *channel*;	チャネル番号 ADCHANNEL0 : 入力チャネル AN000 ADCHANNEL1 : 入力チャネル AN001 ADCHANNEL2 : 入力チャネル AN002 ADCHANNEL3 : 入力チャネル AN003 ADCHANNEL4 : 入力チャネル AN004 ADCHANNEL5 : 入力チャネル AN005 ADCHANNEL6 : 入力チャネル AN006 ADCHANNEL7 : 入力チャネル AN007 ADCHANNEL8 : 入力チャネル AN008 ADCHANNEL9 : 入力チャネル AN009 ADCHANNEL10 : 入力チャネル AN010 ADCHANNEL11 : 入力チャネル AN011 ADCHANNEL12 : 入力チャネル AN012 ADCHANNEL13 : 入力チャネル AN013 ADCHANNEL14 : 入力チャネル AN014 ADCHANNEL15 : 入力チャネル AN015 ADCHANNEL16 : 入力チャネル AN016 ADCHANNEL17 : 入力チャネル AN017 ADCHANNEL18 : 入力チャネル AN018 ADCHANNEL19 : 入力チャネル AN019 ADCHANNEL20 : 入力チャネル AN020 ADTEMPSENSOR : 拡張アナログ入力（温度センサ出力） ADINTERREFVOLT : 拡張アナログ入力（内部基準電圧） ADSELFDIAGNOSIS : 自己診断結果 ADDATADUPLICATION : ダブルトリガモード結果 ADDATADUPLICATIONA : ダブルトリガモード A 結果 ADDATADUPLICATIONB : ダブルトリガモード B 結果
O	uint16_t * const *buffer*;	獲得した変換結果を格納する領域へのポインタ

[戻り値]

なし

R_<Config_S12AD0>_Set_CompareValue

コンペアレベルの設定を行います。

[指定形式]

void R_<Config_S12AD0>_Set_CompareValue (uint16_t *reg_value0*, uint16_t *reg_value1*);

[引数]

I/O	引数	説明
I	uint16_t *reg_value0*;	コンペアレジスタ 0 に設定する値
I	uint16_t *reg_value1*;	コンペアレジスタ 1 に設定する値

[戻り値]

なし

R_<Config_S12AD0>_Set_CompareAValue

ウィンドウ A コンペアレベルの設定を行います。

[指定形式]

void R_<Config_S12AD0>_Set_CompareAValue (uint16_t *reg_value0*, uint16_t *reg_value1*);

[引数]

I/O	引数	説明
I	uint16_t *reg_value0*;	コンペアレジスタ 0 に設定する値
I	uint16_t *reg_value1*;	コンペアレジスタ 1 に設定する値

[戻り値]

なし

R_<Config_S12AD0>_Set_CompareBValue

ウィンドウ B コンペアレベルの設定を行います。

[指定形式]

void R_<Config_S12AD0>_Set_CompareBValue (uint16_t *reg_value0*, uint16_t *reg_value1*);

[引数]

I/O	引数	説明
I	uint16_t *reg_value0*;	コンペアレジスタ 0 に設定する値
I	uint16_t *reg_value1*;	コンペアレジスタ 1 に設定する値

[戻り値]

なし

R_<Config_S12AD0>_Create_UserInit

グループスキャンモード S12AD に関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_S12AD0>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void    R_<Config_S12AD0>_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_S12AD0>_interrupt

スキャン終了割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_S12AD0>_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_S12AD0>_compare_interrupt

コンペア割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_S12AD0>_compare_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_S12AD0>_compare_interruptA

ウィンドウ A コンペア割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_S12AD0>_compare_interruptA ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_S12AD0>_compare_interruptB

ウィンドウ B コンペア割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_S12AD0>_compare_interruptB ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_S12AD0>_groupb_interrupt

グループ B スキャン終了割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_S12AD0>_groupb_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_S12AD0>_groupc_interrupt

グループ C スキャン終了割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_S12AD0>_groupc_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

使用例

グループ A と B の AD 変換結果を取得する

main.c

```
#include "r_smc_entry.h"
void main(void)
{
    /* Start the AD0 converter */
    R_Config_S12AD0_Start();

    while (1U)
    {
        nop();
    }
}
```

Config_S12AD0_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint16_t g_s12ad0_ch000_value;
volatile uint16_t g_s12ad0_ch001_value;
/* End user code. Do not edit comment generated here */

static void r_Config_S12AD0_interrupt(void)
{
    /* Start user code for r_Config_S12AD0_interrupt. Do not edit comment generated here */
    /* Get result from the AD0 channel 0 (AN000) converter */
    R_Config_S12AD0_Get_ValueResult(ADCHANNEL0, (uint16_t *)&g_s12ad0_ch000_value);
    /* End user code. Do not edit comment generated here */
}

static void r_Config_S12AD0_groupb_interrupt(void)
{
    /* Start user code for r_Config_S12AD0_groupb_interrupt. Do not edit comment generated here */
    /* Get result from the AD0 channel 1 (AN001) converter */
    R_Config_S12AD0_Get_ValueResult(ADCHANNEL1, (uint16_t *)&g_s12ad0_ch001_value);
    /* End user code. Do not edit comment generated here */
}
```

4.2.18 I2C マスタモード

以下に、コード生成ツールが I2C マスタモード（RIIC/SCI/RSCI）用として出力する API 関数の一覧を示します。

表 4.18 I2C マスタモード用 API 関数 (RIIC)

API 関数名	機能概要
R_<Config_RIIC0>_Create	I2C マスタモードを制御するうえで必要となる初期化処理を行います。
R_<Config_RIIC0>_Start	通信を開始します。
R_<Config_RIIC0>_Stop	通信を終了します。
R_<Config_RIIC0>_Master_Send	マスタ送信を開始します。
R_<Config_RIIC0>_Master_Send_Without_Stop	マスタ送信を開始します。 （送信終了時にストップコンディションを発行しない）
R_<Config_RIIC0>_Master_Receive	マスタ受信を開始します。
R_<Config_RIIC0>_IIC_StartCondition	スタートコンディションを発行します。
R_<Config_RIIC0>_IIC_StopCondition	ストップコンディションを発行します。
R_<Config_RIIC0>_Create_UserInit	I2C マスタモードに関するユーザ独自の初期化処理を行います。
r_<Config_RIIC0>_error_interrupt	通信エラー／通信イベント発生割り込みの発生に伴う処理を行います。
r_<Config_RIIC0>_receive_interrupt	受信データフル割り込みの発生に伴う処理を行います。
r_<Config_RIIC0>_transmit_interrupt	送信データエンプティ割り込みの発生に伴う処理を行います。
r_<Config_RIIC0>_transmitend_interrupt	送信終了割り込みの発生に伴う処理を行います。
r_<Config_RIIC0>_callback_error	通信エラー／通信イベント発生割り込みに対応した割り込み処理のうち、アービトレーションロスト検出、NACK 検出、タイムアウト検出、通信シーケンスエラー検出に特化した処理を行います。
r_<Config_RIIC0>_callback_transmitend	通信エラー／通信イベント発生割り込みに対応した割り込み処理のうち、マスタ送信に伴うストップコンディションの検出に特化した処理を行います。 また、ストップコンディションを発行しないマスタ送信では送信終了割り込みの処理を行います。
r_<Config_RIIC0>_callback_receiveend	通信エラー／通信イベント発生割り込みに対応した割り込み処理のうち、マスタ受信に伴うストップコンディションの検出に特化した処理を行います。

表 4.19 I2C マスタモード用 API 関数 (SCI/RSCI シンプル I2C モード)

API 関数名	機能概要
R_<Config_SCI0>_Create	I2C マスタモードを制御するうえで必要となる初期化処理を行います。
R_<Config_SCI0>_Start	通信を開始します。
R_<Config_SCI0>_Stop	通信を終了します。
R_<Config_SCI0>_IIC_Master_Send	マスタ送信を開始します。
R_<Config_SCI0>_IIC_Master_Receive	マスタ受信を開始します。
R_<Config_SCI0>_IIC_StartCondition	スタートコンディションを発行します。
R_<Config_SCI0>_IIC_StopCondition	ストップコンディションを発行します。
R_<Config_SCI0>_Create_UserInit	I2C マスタモードに関するユーザ独自の初期化処理を行います。
r_<Config_SCI0>_receive_interrupt	受信データフル割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_transmit_interrupt	送信データエンプティ割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_transmitend_interrupt	開始条件／再開始条件／停止条件生成終了割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_callback_transmitend	開始条件／再開始条件／停止条件生成終了割り込みに対応した割り込み処理のうち、マスタ送信に伴うストップコンディションの検出に特化した処理を行います。 また、送信データエンプティ割り込みを DTC または DMAC の起動要因に設定している際には割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_callback_receiveend	開始条件／再開始条件／停止条件生成終了割り込みに対応した割り込み処理のうち、マスタ受信に伴うストップコンディションの検出に特化した処理を行います。 また、受信データフル割り込みを DTC または DMAC の起動要因に設定している際には割り込みの発生に伴う処理を行います。

R_<Config_RIIC0>_Create

I2C マスタモードを制御するうえで必要となる初期化処理を行います。

備考 本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void R_<Config_RIIC0>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_RIIC0>_Start

通信を開始します。

[指定形式]

```
void        R_<Config_RIIC0>_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_RIIC0>_Stop

通信を終了します。

[指定形式]

```
void        R_<Config_RIIC0>_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_RIIC0>_Master_Send

マスタ送信を開始します。

備考 1. 本 API 関数では、データ（引数 *adr* で指定されたスレーブアドレスと R/W# ビット）をスレーブデバイスにマスタ送信したのち、引数 *tx_buf* で指定されたバッファから 1 バイト単位のマスタ送信を引数 *tx_num* で指定された回数だけ繰り返し行います。

備考 2. 本 API 関数では、マスタ送信の開始処理として、内部的に R_<Config_RIIC0>_IIC_StartCondition の呼び出しを行っています。

備考 3. マスタ送信の終了処理として、r_<Config_RIIC0>_transmitend_interrupt にてストップコンディションを発行しています。

備考 4. マスタ送信を行う際には、本 API 関数の呼び出し以前に R_<Config_RIIC0>_Start を呼び出す必要があります。

[指定形式]

```
MD_STATUS  R_<Config_RIIC0>_Master_Send ( uint16_t adr, uint8_t * const tx_buf, uint16_t tx_num );
```

[引数]

I/O	引数	説明
I	uint16_t *adr*;	スレーブアドレス 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 スレーブアドレス(0～1023)
I	uint8_t * const *tx_buf*;	送信するデータを格納したバッファへのポインタ
I	uint16_t *tx_num*;	送信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR1	バスビジー
MD_ERROR2	引数 *adr* の指定が不正

R_<Config_RIIC0>_Master_Send_Without_Stop

【 RIIC 】マスタ送信を開始します。(送信終了時にストップコンディションを発行しない)

備考 1. 本 API 関数では、データ（引数 *adr* で指定されたスレーブアドレスと R/W# ビット）をスレーブデバイスにマスタ送信したのち、引数 *tx_buf* で指定されたバッファから 1 バイト単位のマスタ送信を引数 *tx_num* で指定された回数だけ繰り返し行います。

備考 2. 本 API 関数では、マスタ送信の開始処理として、内部的に R_<Config_RIIC0>_IIC_StartCondition の呼び出しを行っています。

備考 3. マスタ送信の終了処理として、r_<Config_RIIC0>_transmitend_interrupt にてストップコンディションを発行しません。

備考 4. マスタ送信を行う際には、本 API 関数の呼び出し以前に R_<Config_RIIC0>_Start を呼び出す必要があります。

[指定形式]

```
MD_STATUS      R_<Config_RIIC0>_Master_Send_Without_Stop ( uint16_t adr, uint8_t * const tx_buf, uint16_t tx_num );
```

[引数]

I/O	引数	説明
I	uint16_t *adr*;	スレーブアドレス 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 スレーブアドレス(0～1023)
I	uint8_t * const *tx_buf*;	送信するデータを格納したバッファへのポインタ
I	uint16_t *tx_num*;	送信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR1	バスビジー
MD_ERROR2	引数 *adr* の指定が不正

R_<Config_RIIC0>_Master_Receive

【 RIIC 】マスタ受信を開始します。

備考 1. 本 API 関数では、データ（引数 *adr* で指定されたスレーブアドレスと R/W# ビット）をスレーブデバイスにマスタ送信したのち、 1 バイト単位のマスタ受信を引数 *rx_num* で指定された回数だけ繰り返し行い、引数 *rx_buf* で指定されたバッファに格納します。

備考 2. 本 API 関数では、マスタ受信の開始処理として、内部的に R_<Config_RIIC0>_IIC_StartCondition の呼び出しを行っています。

備考 3. マスタ受信の終了処理として、r_<Config_RIIC0>_receive_interrupt にてストップコンディションを発行しています。

備考 4. マスタ受信を行う際には、本 API 関数の呼び出し以前に R_<Config_RIIC0>_Start を呼び出す必要があります。

[指定形式]

```
MD_STATUS    R_<Config_RIIC0>_Master_Receive ( uint16_t adr, uint8_t * const rx_buf, uint16_t rx_num );
```

[引数]

I/O	引数	説明
I	uint16_t *adr*;	スレーブアドレス 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 スレーブアドレス(0～127)
O	uint8_t * const *rx_buf*;	受信したデータを格納するバッファへのポインタ
I	uint16_t *rx_num*;	受信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR2	引数 *adr* の指定が不正
MD_ERROR3	引数 *adr* の指定が不正（マスタ受信は 10bit アドレス非対応です）
MD_ERROR4	バスビジー（タイムアウト検出，アービトレーションロスト検出）
MD_ERROR5	バスビジー（ストップコンディション未検出）

R_<Config_RIIC0>_IIC_StartCondition

スタートコンディションを発行します。

備考　本 API 関数の呼び出しに伴い、通信エラー／通信イベント発生割り込みを発生させ、r_<Config_RIIC0>_error_interrupt が呼び出されます。

[指定形式]

void	R_<Config_RIIC0>_IIC_StartCondition (void);

[引数]

なし

[戻り値]

なし

R_<Config_RIIC0>_IIC_StopCondition

ストップコンディションを発行します。

備考　本 API 関数の呼び出しに伴い、通信エラー／通信イベント発生割り込みを発生させ、r_<Config_RIIC0>_error_interrupt が呼び出されます。

[指定形式]

void	R_<Config_RIIC0>_IIC_StopCondition (void);

[引数]

なし

[戻り値]

なし

R_<Config_RIIC0>_Create_UserInit

I2C マスタモードに関するユーザ独自の初期化処理を行います。

備考 本 API 関数は、R_<Config_RIIC0>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_RIIC0>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

r_<Config_RIIC0>_error_interrupt

通信エラー／通信イベント発生割り込みの発生に伴う処理を行います。

備考　本 API 関数は、通信エラー／通信イベント発生（アービトレーションロスト検出、NACK 検出、タイムアウト検出、スタートコンディション検出、ストップコンディション検出）に対応した割り込み処理として呼び出されます。

[指定形式]

```
void    r_<Config_RIIC0>_error_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_RIIC0>_receive_interrupt

受信データフル割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_RIIC0>_receive_interrupt (void);

[引数]

なし

[戻り値]

なし

r_<Config_RIIC0>_transmit_interrupt

送信データエンプティ割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_RIIC0>_transmit_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_RIIC0>_transmitend_interrupt

送信終了割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_RIIC0>_transmitend_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_RIIC0>_callback_error

通信エラー／通信イベント発生割り込みに対応した割り込み処理のうち、アービトレーションロスト検出、NACK 検出、タイムアウト検出、通信シーケンスエラー検出に特化した処理を行います。

備考　本 API 関数は、r_<Config_RIIC0>_error_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	r_<Config_RIIC0>_callback_error (MD_STATUS *status*);

[引数]

I/O	引数	説明
I	MD_STATUS *status*;	割り込みの発生要因 MD_ERROR1：アービトレーションロスト検出 MD_ERROR2：タイムアウト検出 MD_ERROR3：NACK 検出 MD_ERROR4：通信シーケンスエラー検出

[戻り値]

なし

r_<Config_RIIC0>_callback_transmitend

通信エラー／通信イベント発生割り込みに対応した割り込み処理のうち、マスタ送信に伴うストップコンディションの検出に特化した処理を行います。

備考 1. 本 API 関数は、r_<Config_RIIC0>_error_interrupt のコールバック・ルーチンとして呼び出されます。

備考 2. マスタ送信を行う際には、R_<Config_RIIC0>_Master_Send を呼び出してください。

また、ストップコンディションを発行しないマスタ送信では送信終了割り込みの処理を行います。

備考 3. 本 API 関数は、r_<Config_RIIC0>_transmitend_interrupt のコールバック・ルーチンとして呼び出されます。

備考 4. マスタ送信を行う際には、R_<Config_RIIC0>_Master_Send_Without_Stop を呼び出してください。

[指定形式]

```
void        r_<Config_RIIC0>_callback_transmitend ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_RIIC0>_callback_receiveend

通信エラー／通信イベント発生割り込みに対応した割り込み処理のうち、マスタ受信に伴うストップコンディションの検出に特化した処理を行います。

備考 1. 本 API 関数は、r_<Config_RIIC0>_error_interrupt のコールバック・ルーチンとして呼び出されます。

備考 2. マスタ受信を行う際には、R_<Config_RIIC0>_Master_Receive を呼び出してください。

[指定形式]

void	r_<Config_RIIC0>_callback_receiveend (void);

[引数]

なし

[戻り値]

なし

R_<Config_SCI0>_Create

I2C マスタモードを制御するうえで必要となる初期化処理を行います。

備考 本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void R_<Config_SCI0>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_SCI0>_Start

通信を開始します。

[指定形式]

```
void        R_<Config_SCI0>_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_SCI0>_Stop

通信を終了します。

[指定形式]

```
void        R_<Config_SCI0>_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_SCI0>_IIC_Master_Send

【 SCI 】マスタ送信を開始します。(簡易 I2C モード)

備考 1. 本 API 関数では、データ(引数 *adr* で指定されたスレーブアドレスと R/W# ビット)をスレーブデバイスにマスタ送信したのち、引数 *tx_buf* で指定されたバッファから 1 バイト単位のマスタ送信を引数 *tx_num* で指定された回数だけ繰り返し行います。

備考 2. 本 API 関数では、マスタ送信の開始処理として、内部的に R_<Config_SCI0>_IIC_StartCondition の呼び出しを行っています。

備考 3. マスタ送信の終了処理として、r_<Config_SCI0>_transmit_interrupt にてストップコンディションを発行しています。

備考 4. マスタ送信を行う際には、本 API 関数の呼び出し以前に R_<Config_SCI0>_Start を呼び出す必要があります。

[指定形式]

```
void    R_<Config_SCI0>_IIC_Master_Send ( uint8_t adr, uint8_t * const tx_buf, uint16_t tx_num );
```

[引数]

I/O	引数	説明
I	uint8_t *adr*;	スレーブアドレス 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 スレーブアドレス(0～127) | W(0)
I	uint8_t * const *tx_buf*;	送信するデータを格納したバッファへのポインタ
I	uint16_t *tx_num*;	送信するデータの総数

[戻り値]

なし

R_<Config_SCI0>_IIC_Master_Receive

【 SCI 】マスタ受信を開始します。（簡易 I2C モード）

備考 1. 本 API 関数では、データ（引数 *adr* で指定されたスレーブアドレス）をスレーブデバイスにマスタ送信したのち、 1 バイト単位のマスタ受信を引数 *rx_num* で指定された回数だけ繰り返し行い、引数 *rx_buf* で指定されたバッファに格納します。

備考 2. 本 API 関数では、マスタ受信の開始処理として、内部的に R_<Config_SCI0>_IIC_StartCondition の呼び出しを行っています。

備考 3. マスタ受信の終了処理として、r_<Config_SCI0>_receive_interrupt にてストップコンディションを発行しています。

備考 4. マスタ受信を行う際には、本 API 関数の呼び出し以前に R_<Config_SCI0>_Start を呼び出す必要があります。

[指定形式]

```
void    R_<Config_SCI0>_IIC_Master_Receive ( uint8_t adr, uint8_t * const rx_buf, uint16_t rx_num );
```

[引数]

I/O	引数	説明
I	uint8_t *adr*;	スレーブアドレス 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 スレーブアドレス(0～127) | R(1)
O	uint8_t * const *rx_buf*;	受信したデータを格納するバッファへのポインタ
I	uint16_t *rx_num*;	受信するデータの総数

[戻り値]

なし

R_<Config_SCI0>_IIC_StartCondition

スタートコンディションを発行します。

備考　本 API 関数の呼び出しに伴い、開始条件／再開始条件／停止条件生成終了割り込みを発生させ、r_<Config_SCI0>_transmitend_interrupt が呼び出されます。

[指定形式]

```
void    R_<Config_SCI0>_IIC_StartCondition ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_SCI0>_IIC_StopCondition

ストップコンディションを発行します。

備考　本 API 関数の呼び出しに伴い、開始条件／再開始条件／停止条件生成終了割り込みを発生させ、r_<Config_SCI0>_transmitend_interrupt が呼び出されます。

[指定形式]

void	R_<Config_SCI0>_IIC_StopCondition (void);

[引数]

なし

[戻り値]

なし

R_<Config_SCI0>_Create_UserInit

I2C マスタモードに関するユーザ独自の初期化処理を行います。

備考 本 API 関数は、R_<Config_SCI0>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_SCI0>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_receive_interrupt

受信データフル割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_SCI0>_receive_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_transmit_interrupt

送信データエンプティ割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_SCI0>_transmit_interrupt (void);

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_transmitend_interrupt

開始条件／再開始条件／停止条件生成終了割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_SCI0>_transmitend_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_callback_transmitend

開始条件／再開始条件／停止条件生成終了割り込みに対応した割り込み処理のうち、マスタ送信に伴うストップコンディションの検出に特化した処理を行います。

備考 1. 本 API 関数は、r_<Config_SCI0>_transmitend_interrupt のコールバック・ルーチンとして呼び出されます。

備考 2. マスタ送信を行う際には、R_<Config_SCI0>_IIC_Master_Send を呼び出してください。

また、送信データエンプティ割り込みを DTC または DMAC の起動要因に設定している際には割り込みの発生に伴う処理を行います。

備考 3. 本 API 関数は、r_<Config_SCI0>_transmit_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void        r_<Config_SCI0>_callback_transmitend ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_callback_receiveend

開始条件／再開始条件／停止条件生成終了割り込みに対応した割り込み処理のうち、マスタ受信に伴うストップコンディションの検出に特化した処理を行います。

備考 3. 本 API 関数は、r_<Config_SCI0>_transmitend_interrupt のコールバック・ルーチンとして呼び出されます。

備考 4. マスタ受信を行う際には、R_<Config_SCI0>_IIC_Master_Receive を呼び出してください。

また、受信データフル割り込みを DTC または DMAC の起動要因に設定している際には割り込みの発生に伴う処理を行います。

備考 5. 本 API 関数は、r_<Config_SCI0>_receive_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void        r_<Config_SCI0>_callback_receiveend ( void );
```

[引数]

なし

[戻り値]

なし

使用例 1

RIIC で、４回のマスタ送信を行う

main.c

```
#include "r_smc_entry.h"
extern volatile uint8_t g_riic0_tx_buf[2];
void main(void)
{
    /* Start the RIIC0 I2C Bus Interface */
    R_Config_RIIC0_Start();

    /* Send RIIC0 data to slave device [Slave address : 160 (10-bit address mode)] */
    R_Config_RIIC0_Master_Send(0x00A0, (uint8_t *)g_riic0_tx_buf, 2U);

    while (1U)
    {
        nop();
    }
}
```

Config_RIIC0_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_riic0_tx_buf[2];
volatile uint8_t g_riic0_tx_cnt;
/* End user code. Do not edit comment generated here */

void R_Config_RIIC0_Create_UserInit(void)
{
    /* Start user code for user init. Do not edit comment generated here */
    g_riic0_tx_cnt = 0U;
    g_riic0_tx_buf[0] = g_riic0_tx_cnt;
    g_riic0_tx_buf[1] = 0x01;
    /* End user code. Do not edit comment generated here */
}

static void r_Config_RIIC0_callback_transmitend(void)
{
    /* Start user code for r_Config_RIIC0_callback_transmitend. Do not edit comment generated
here */
    if ((++g_riic0_tx_cnt) < 4U)
    {
        g_riic0_tx_buf[0] = g_riic0_tx_cnt;
        g_riic0_tx_buf[1] += 0x01;

        /* Send RIIC0 data to slave device [Slave assress : 160 (10-bit address mode)] */
        R_Config_RIIC0_Master_Send(0x00A0, (uint8_t *)g_riic0_tx_buf, 2U);
    }
    else
    {
        /* Stop the RIIC0 I2C Bus Interface */
        R_Config_RIIC0_Stop();
    }
    /* End user code. Do not edit comment generated here */
}
```

使用例 2

SCI 簡易 I2C モードで、4 回のマスタ送信を行う

main.c

```
#include "r_smc_entry.h"
extern volatile uint8_t g_riic0_tx_buf[2];
void main(void)
{
    /* Start the SCI0 I2C Bus Interface */
    R_Config_SCI0_Start();

    /* Send SCI0 data to slave device [Slave address : 80] */
    R_Config_SCI0_Master_Send(0xA0, (uint8_t *)g_sci0_tx_buf, 2U);

    while (1U)
    {
        nop();
    }
}
```

Config_SCI0_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_sci0_tx_buf[2];
volatile uint8_t g_sci0_tx_cnt;
/* End user code. Do not edit comment generated here */

void R_Config_SCI0_Create_UserInit(void)
{
    /* Start user code for user init. Do not edit comment generated here */
    g_sci0_tx_cnt = 0U;
    g_sci0_tx_buf[0] = g_sci0_tx_cnt;
    g_sco0_tx_buf[1] = 0x01;
    /* End user code. Do not edit comment generated here */
}

static void r_Config_SCI0_callback_transmitend(void)
{
    /* Start user code for r_Config_SCI0_callback_transmitend. Do not edit comment generated
here */
    if ((++g_sci0_tx_cnt) < 4U)
    {
        g_sci0_tx_buf[0] = g_sci0_tx_cnt;
        g_sci0_tx_buf[1] += 0x01;

        /* Send SCI0 data to slave device [Slave assress : 80] */
        R_Config_SCI0_Master_Send(0xA0, (uint8_t *)g_sci0_tx_buf, 2U);
    }
    else
    {
        /* Stop the SCI0 I2C Bus Interface */
        R_Config_SCI0_Stop();
    }
    /* End user code. Do not edit comment generated here */
}
```

4.2.19 I2C スレーブモード

以下に、コード生成ツールが I2C スレーブモード用として出力する API 関数の一覧を示します。

表 4.20 I2C スレーブモード用 API 関数

API 関数名	機能概要
R_<Config_RIIC0>_Create	I2C スレーブモードを制御するうえで必要となる初期化処理を行います。
R_<Config_RIIC0>_Start	通信を開始します。
R_<Config_RIIC0>_Stop	通信を終了します。
R_<Config_RIIC0>_Slave_Send	スレーブ送信を開始します。
R_<Config_RIIC0>_Slave_Receive	スレーブ受信を開始します。
R_<Config_RIIC0>_Create_UserInit	I2C スレーブモードに関するユーザ独自の初期化処理を行います。
r_<Config_RIIC0>_error_interrupt	通信エラー／通信イベント発生割り込みの発生に伴う処理を行います。
r_<Config_RIIC0>_receive_interrupt	受信データフル割り込みの発生に伴う処理を行います。
r_<Config_RIIC0>_transmit_interrupt	送信データエンプティ割り込みの発生に伴う処理を行います。
r_<Config_RIIC0>_transmitend_interrupt	送信終了割り込みの発生に伴う処理を行います。
r_<Config_RIIC0>_callback_error	通信エラー／通信イベント発生割り込みに対応した割り込み処理のうち、アービトレーションロスト検出、NACK 検出、タイムアウト検出、通信シーケンスエラー検出に特化した処理を行います。
r_<Config_RIIC0>_callback_transmitend	通信エラー／通信イベント発生割り込みに対応した割り込み処理のうち、スレーブ送信に伴うストップコンディションの検出に特化した処理を行います。
r_<Config_RIIC0>_callback_receiveend	通信エラー／通信イベント発生割り込みに対応した割り込み処理のうち、スレーブ受信に伴うストップコンディションの検出に特化した処理を行います。

R_<Config_RIIC0>_Create

I2C スレーブモードを制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void    R_<Config_RIIC0>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_RIIC0>_Start

通信を開始します。

[指定形式]

```
void        R_<Config_RIIC0>_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_RIIC0>_Stop

通信を終了します。

[指定形式]

```
void        R_<Config_RIIC0>_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_RIIC0>_Slave_Send

スレーブ送信を開始します。

備考 1. 本 API 関数では、引数 *tx_buf* で指定されたバッファから 1 バイト単位のスレーブ送信を引数 *tx_num* で指定された回数だけ繰り返し行います。

備考 2. スレーブ送信を行う際には、本 API 関数の呼び出し以前に R_<Config_RIIC0>_Start を呼び出す必要があります。

[指定形式]

MD_STATUS	R_<Config_RIIC0>_Slave_Send (uint8_t * const *tx_buf*, uint16_t *tx_num*);

[引数]

I/O	引数	説明
I	uint8_t * const *tx_buf*;	送信するデータを格納したバッファへのポインタ
I	uint16_t *tx_num*;	送信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了

R_<Config_RIIC0>_Slave_Receive

スレーブ受信を開始します。

備考 1. 本 API 関数では、 1 バイト単位のスレーブ受信を引数 *rx_num* で指定された回数だけ繰り返し行い、引数 *rx_buf* で指定されたバッファに格納します。

備考 2. スレーブ受信を行う際には、本 API 関数の呼び出し以前に R_<Config_RIIC0>_Start を呼び出す必要があります。

[指定形式]

```
MD_STATUS      R_<Config_RIIC0>_Slave_Receive ( uint8_t * const rx_buf, uint16_t rx_num );
```

[引数]

I/O	引数	説明
O	uint8_t * const *rx_buf*;	受信したデータを格納するバッファへのポインタ
I	uint16_t *rx_num*;	受信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了

R_<Config_RIIC0>_Create_UserInit

I2C スレーブモードに関するユーザ独自の初期化処理を行います。

備考 本 API 関数は、R_<Config_RIIC0>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_RIIC0>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

r_<Config_RIIC0>_error_interrupt

通信エラー／通信イベント発生割り込みの発生に伴う処理を行います。

備考　本 API 関数は、通信エラー／通信イベント発生（アービトレーションロスト検出、NACK 検出、タイムアウト検出、スタートコンディション検出、ストップコンディション検出）に対応した割り込み処理として呼び出されます。

[指定形式]

```
void    r_<Config_RIIC0>_error_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_RIIC0>_receive_interrupt

受信データフル割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_RIIC0>_receive_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_RIIC0>_transmit_interrupt

送信データエンプティ割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_RIIC0>_transmit_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_RIIC0>_transmitend_interrupt

送信終了割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_RIIC0>_transmitend_interrupt (void);

[引数]

なし

[戻り値]

なし

r_<Config_RIIC0>_callback_error

通信エラー／通信イベント発生割り込みに対応した割り込み処理のうち、アービトレーションロスト検出、NACK 検出、タイムアウト検出、通信シーケンスエラー検出に特化した処理を行います。

備考　本 API 関数は、r_<Config_RIIC0>_error_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	r_<Config_RIIC0>_callback_error (MD_STATUS *status*);

[引数]

I/O	引数	説明
I	MD_STATUS *status*;	割り込みの発生要因 MD_ERROR1：アービトレーションロスト検出 MD_ERROR2：タイムアウト検出 MD_ERROR3：NACK 検出 MD_ERROR4：通信シーケンスエラー検出

[戻り値]

なし

r_<Config_RIIC0>_callback_transmitend

通信エラー／通信イベント発生割り込みに対応した割り込み処理のうち、スレーブ送信に伴うストップコンディションの検出に特化した処理を行います。

備考 1. 本 API 関数は、r_<Config_RIIC0>_error_interrupt のコールバック・ルーチンとして呼び出されます。

備考 2. スレーブ送信を行う際には、R_<Config_RIIC0>_Slave_Send を呼び出してください。

[指定形式]

```
void        r_<Config_RIIC0>_callback_transmitend ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_RIIC0>_callback_receiveend

通信エラー／通信イベント発生割り込みに対応した割り込み処理のうち、スレーブ受信に伴うストップコンディションの検出に特化した処理を行います。

備考 1. 本 API 関数は、r_<Config_RIIC0>_error_interrupt のコールバック・ルーチンとして呼び出されます。

備考 2. スレーブ受信を行う際には、R_<Config_RIIC0>_Slave_Receive を呼び出してください。

[指定形式]

void	r_<Config_RIIC0>_callback_receiveend (void);

[引数]

なし

[戻り値]

なし

使用例

４回のスレーブ受信を行う

main.c

```
#include "r_smc_entry.h"
extern volatile uint8_t g_riic2_rx_buf[2];
void main(void)
{
    /* Start the RIIC2 Bus Interface */
    R_Config_RIIC2_Start();

    /* Read data from a master device */
    R_Config_RIIC2_Slave_Receive((uint8_t *)g_riic2_rx_buf, 2U);

    while (1U)
    {
        nop();
    }
}
```

Config_RIIC2_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_riic2_rx_buf[2];
volatile uint8_t g_riic2_rx_cnt;
/* End user code. Do not edit comment generated here */

void R_Config_RIIC2_Create_UserInit(void)
{
    /* Start user code for user init. Do not edit comment generated here */
    g_riic2_rx_cnt = 0U;
    /* End user code. Do not edit comment generated here */
}

static void r_Config_RIIC2_callback_receiveend(void)
{
    /* Start user code for r_Config_RIIC2_callback_receiveend. Do not edit comment generated
here */
    if ((++g_riic2_rx_cnt) < 4U)
    {
        /* Read data from a master device */
        R_Config_RIIC2_Slave_Receive((uint8_t *)g_riic2_rx_buf, 2U);
    }
    else
    {
        /* Stop the RIIC2 Bus Interface */
        R_Config_RIIC2_Stop();
    }
    /* End user code. Do not edit comment generated here */
}
```

4.2.20 割り込みコントローラ

以下に、コード生成ツールが割り込みコントローラ用として出力する API 関数の一覧を示します。

表 4.21 割り込みコントローラ用 API 関数

API 関数名	機能概要
R_<Config_ICU>_Create	割り込みコントローラを制御するうえで必要となる初期化処理を行います。
R_<Config_ICU>_IRQn_Start	外部端子割り込みの検出を許可します。
R_<Config_ICU>_IRQn_Stop	外部端子割り込みの検出を禁止します。
R_<Config_ICU>_Software_Start	ソフトウェア割り込みの検出を許可します。
R_<Config_ICU>_Software_Stop	ソフトウェア割り込みの検出を禁止します。
R_<Config_ICU>_SoftwareInterrupt_Generate	ソフトウェア割り込みを発生させます。
R_<Config_ICU>_Software2_Start	ソフトウェア割り込み 2 の検出を許可します。
R_<Config_ICU>_Software2_Stop	ソフトウェア割り込み 2 の検出を禁止します。
R_<Config_ICU>_SoftwareInterrupt2_Generate	ソフトウェア割り込み 2 を発生させます。
R_<Config_ICU>_Create_UserInit	割り込みコントローラに関するユーザ独自の初期化処理を行います。
r_<Config_ICU>_irqn_interrupt	外部端子割り込みの発生に伴う処理を行います。
r_<Config_ICU>_software_interrupt	ソフトウェア割り込みの発生に伴う処理を行います。
r_<Config_ICU>_software2_interrupt	ソフトウェア割り込み 2 の発生に伴う処理を行います。
r_<Config_ICU>_nmi_interrupt	NMI 端子割り込みの発生に伴う処理を行います。

R_<Config_ICU>_Create

割り込みコントローラを制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_<Config_ICU>_IRQ*n*_Start

外部端子割り込みの検出を許可します。

[指定形式]

void	R_<Config_ICU>_IRQ*n*_Start (void);

備考 *n* は IRQ 端子番号を意味します。

[引数]

なし

[戻り値]

なし

R_<Config_ICU>_IRQ*n*_Stop

外部端子割り込みの検出を禁止します。

[指定形式]

void	R_<Config_ICU>_IRQ*n*_Stop (void);

備考 *n* は IRQ 端子番号を意味します。

[引数]

なし

[戻り値]

なし

R_<Config_ICU>_Software_Start

ソフトウェア割り込みの検出を許可します。

[指定形式]

```
void        R_<Config_ICU>_Software_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_ICU>_Software_Stop

ソフトウェア割り込みの検出を禁止します。

[指定形式]

void	R_<Config_ICU>_Software_Stop (void);

[引数]

なし

[戻り値]

なし

R_<Config_ICU>_SoftwareInterrupt_Generate

ソフトウェア割り込みを発生させます。

備考 本 API 関数の呼び出しに伴い、r_<Config_ICU>_software_interrupt が呼び出されます。

[指定形式]

void	R_<Config_ICU>_SoftwareInterrupt_Generate (void);

[引数]

なし

[戻り値]

なし

R_<Config_ICU>_Software2_Start

ソフトウェア割り込み 2 の検出を許可します。

[指定形式]

void	R_<Config_ICU>_Software2_Start (void);

[引数]

なし

[戻り値]

なし

R_<Config_ICU>_Software2_Stop

ソフトウェア割り込み 2 の検出を禁止します。

[指定形式]

```
void        R_<Config_ICU>_Software2_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_ICU>_SoftwareInterrupt2_Generate

ソフトウェア割り込み 2 を発生させます。

備考 本 API 関数の呼び出しに伴い、r_<Config_ICU>_software2_interrupt が呼び出されます。

[指定形式]

void	R_<Config_ICU>_SoftwareInterrupt2_Generate (void);

[引数]

なし

[戻り値]

なし

R_<Config_ICU>_Create_UserInit

割り込みコントローラに関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_ICU>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_ICU>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

r_<Config_ICU>_irq*n*_interrupt

外部端子割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_ICU>_irq*n*_interrupt (void);

備考 *n* は IRQ 端子番号を意味します。

[引数]

なし

[戻り値]

なし

r_<Config_ICU>_software_interrupt

ソフトウェア割り込みの発生に伴う処理を行います。

備考 本 API 関数は、R_<Config_ICU>_SoftwareInterrupt_Generate を呼び出した場合に発生するソフトウェア割り込みに対応した割り込み処理として呼び出されます。

[指定形式]

```
void        r_<Config_ICU>_software_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_ICU>_software2_interrupt

ソフトウェア割り込みの発生に伴う処理を行います。

備考 本 API 関数は、R_<Config_ICU>_SoftwareInterrupt2_Generate を呼び出した場合に発生するソフトウェア割り込みに対応した割り込み処理として呼び出されます。

[指定形式]

```
void    r_<Config_ICU>_software2_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_ICU>_nmi_interrupt

NMI 端子割り込みの発生に伴う処理を行います。

[指定形式]

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

[引数]

なし

[戻り値]

なし

使用例

外部割り込みでスリープモードから復帰する

main.c

```
#include "r_smc_entry.h"
void main(void)
{
    /* Enable IRQ0 interrupt */
    R_Config_ICU_IRQ0_Start();

    /* Enable sleep mode */
    R_Config_LPC_Sleep();

    while (1U)
    {
        nop();
    }
}
```

Config_ICU_user.c

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

static void r_Config_ICU_irq0_interrupt(void)
{
    /* Start user code for r_Config_ICU_irq0_interrupt. Do not edit comment generated here */
    /* Allow sleep mode return clock to be changed */
    R_Config_LPC_ChangeSleepModeReturnClock(RETURN_MAIN_CLOCK);
    /* End user code. Do not edit comment generated here */
}
```

4.2.21 消費電力低減機能

以下に、コード生成ツールが消費電力低減機能用として出力する API 関数の一覧を示します。

表 4.22 消費電力低減機能用 API 関数

API 関数名	機能概要
R_<Config_LPC>_Create	消費電力低減機能を制御するうえで必要となる初期化処理を行います。
R_<Config_LPC>_AllModuleClockStop	全モジュールのクロックを停止します。
R_<Config_LPC>_Sleep	MCU の低消費電力状態をスリープモードへと遷移させます。
R_<Config_LPC>_DeepSleep	MCU の低消費電力状態をディープスリープモードへと遷移させます。
R_<Config_LPC>_SoftwareStandby	MCU の低消費電力状態をソフトウェアスタンバイモードへと遷移させます。
R_<Config_LPC>_DeepSoftwareStandby	MCU の低消費電力状態をディープソフトウェアスタンバイモードへと遷移させます。
R_<Config_LPC>_ChangeOperatingPowerControl	MCU の動作電力制御状態を変更します。
R_<Config_LPC>_ChangeSleepModeReturnClock	スリープモードが解除された際に選択されるクロックソースを設定します。
R_<Config_LPC>_Create_UserInit	消費電力低減機能に関するユーザ独自の初期化処理を行います。
R_<Config_LPC>_SetVOLSR_PGAVLS	プログラマブルゲインアンプ動作条件を設定します。

R_<Config_LPC>_Create

消費電力低減機能を制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_<Config_LPC>_AllModuleClockStop

全モジュールのクロックを停止します。

[指定形式]

```
MD_STATUS       R_<Config_LPC>_AllModuleClockStop ( void );
```

[引数]

なし

[戻り値]

マクロ	説明
MD_OK	正常終了

R_<Config_LPC>_Sleep

MCU の低消費電力状態をスリープモードへと遷移させます。

[指定形式]

MD_STATUS	R_<Config_LPC>_Sleep (void);

[引数]

なし

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR1	異常終了

R_<Config_LPC>_DeepSleep

MCU の低消費電力状態をディープスリープモードへと遷移させます。

[指定形式]

```
MD_STATUS        R_<Config_LPC>_DeepSleep ( void );
```

[引数]

なし

[戻り値]

マクロ	説明
MD_OK	正常終了

R_<Config_LPC>_SoftwareStandby

MCU の低消費電力状態をソフトウェアスタンバイモードへと遷移させます。

[指定形式]

MD_STATUS	R_<Config_LPC>_SoftwareStandby (void);

[引数]

なし

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR1	異常終了

R_<Config_LPC>_DeepSoftwareStandby

MCU の低消費電力状態をディープソフトウェアスタンバイモードへと遷移させます。

[指定形式]

MD_STATUS	R_<Config_LPC>_DeepSoftwareStandby (void);

[引数]

なし

[戻り値]

マクロ	説明
MD_OK	正常終了

R_<Config_LPC>_ChangeOperatingPowerControl

MCU の動作電力制御状態を変更します。

[指定形式]

```
MD_STATUS      R_<Config_LPC>_ChangeOperatingPowerControl ( operating_mode_t mode );
```

[引数]

RX130，RX230/RX231 の場合

I/O	引数	説明
I	operating_mode_t *mode*;	動作電力制御状態の種類 HIGH_SPEED ：高速動作モード MIDDLE_SPEED ：中速動作モード LOW_SPEED ：低速動作モード

そのほかのデバイスの場合

I/O	引数	説明
I	operating_mode_t *mode*;	動作電力制御状態の種類 HIGH_SPEED ：高速動作モード LOW_SPEED1 ：低速動作モード 1 LOW_SPEED2 ：低速動作モード 2

[戻り値]

RX130，RX230/RX231 の場合

マクロ	説明
MD_OK	正常終了
MD_ERROR1	低速動作モードへの変更が異常終了
MD_ARGERROR	引数 *mode* の指定が不正

そのほかのデバイスの場合

マクロ	説明
MD_OK	正常終了
MD_ERROR1	低速動作モード 1 への変更が異常終了
MD_ERROR2	低速動作モード 2 への変更が異常終了
MD_ARGERROR	引数 *mode* の指定が不正

R_<Config_LPC>_ChangeSleepModeReturnClock

スリープモードが解除された際に選択されるクロックソースを設定します。

[指定形式]

MD_STATUS	R_<Config_LPC>_ChangeSleepModeReturnClock (return_clock_t *clock*);

[引数]

RX130，RX230/RX231 の場合

I/O	引数	説明
I	return_clock_t *clock*;	クロックソースの種類 RETURN_LOCO ：低速オンチップオシレータ RETURN_HOCO ：高速オンチップオシレータ RETURN_MAIN_CLOCK ：メインクロック発振器 RETURN_DISABLE ：クロックソースの切り替えを行わない

そのほかのデバイスの場合

I/O	引数	説明
I	return_clock_t *clock*;	クロックソースの種類 RETURN_HOCO ：高速オンチップオシレータ RETURN_MAIN_CLOCK ：メインクロック発振器 RETURN_DISABLE ：クロックソースの切り替えを行わない

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR1	現在設定されているクロックソースの指定が不正
MD_ARGERROR	引数 *clock* の指定が不正

R_<Config_LPC>_Create_UserInit

消費電力低減機能に関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_LPC>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_LPC>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

R_<Config_LPC>_SetVOLSR_PGAVLS

プログラマブルゲインアンプ動作条件を設定します。

[指定形式]

```
MD_STATUS   R_<Config_LPC>_SetVOLSR_PGAVLS ( uint8_t pgavls_bit );
```

[引数]

I/O	引数	説明
I	uint8_t pgavls_bit;	プログラマブルゲインアンプ動作条件 0u　：AVCC の電圧が 4.0V 以上、かつ PGA の疑似差動入力を有効にし、端子に負電圧を入力する 1u　：AVCC の電圧が 4.0V 未満、または端子に負電圧を入力しない

[戻り値]

マクロ	説明
MD_OK	正常終了

使用例

割り込みコントローラ使用例を参照

4.2.22 ローパワータイマ

以下に、コード生成ツールがローパワータイマ用として出力する API 関数の一覧を示します。

表 4.23 ローパワータイマ用 API 関数

API 関数名	機能概要
R_<Config_LPT>_Create	ローパワータイマを制御するうえで必要となる初期化処理を行います。
R_<Config_LPT>_Start	カウントを開始します。
R_<Config_LPT>_Stop	カウントを終了します。
R_<Config_LPT>_Create_UserInit	ローパワータイマに関するユーザ独自の初期化処理を行います。

R_<Config_LPT>_Create

ローパワータイマを制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_<Config_LPT>_Start

カウントを開始します。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_<Config_LPT>_Stop

カウントを終了します。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_<Config_LPT>_Create_UserInit

ローパワータイマに関するユーザ独自の初期化処理を行います。

備考 本 API 関数は、R_<Config_LPT>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_LPT>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

使用例

なし

4.2.23 ノーマルモード

以下に、コード生成ツールがノーマルモード（MTU/TPU）用として出力する API 関数の一覧を示します。

表 4.24 ノーマルモード用 API 関数

API 関数名	機能概要
R_<Config_MTU0>_Create	ノーマルモード（MTU/TPU）を制御するうえで必要となる初期化処理を行います。
R_<Config_MTU0>_Start	カウンタを開始します。
R_<Config_MTU0>_Stop	カウンタを終了します。
R_<Config_MTU0>_Create_UserInit	ノーマルモード（MTU/TPU）に関するユーザ独自の初期化処理を行います。
r_<Config_MTU0>_tgimn_interrupt	インプットキャプチャ／コンペアマッチ割り込みの発生に伴う処理を行います。 （リソースにより、API 関数名が異なります。）
r_<Config_MTU0>_tginm_interrupt	
r_<Config_MTU0>_tcivn_interrupt	オーバフロー割り込みの発生に伴う処理を行います。 （リソースにより、API 関数名が異なります。）
r_<Config_MTU0>_tcinv_interrupt	

R_<Config_MTU0>_Create

ノーマルモード（MTU/TPU）を制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void    R_<Config_MTU0>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_MTU0>_Start

カウンタを開始します。

[指定形式]

void	R_<Config_MTU0>_Start (void);

[引数]
なし

[戻り値]
なし

R_<Config_MTU0>_Stop

カウンタを終了します。

[指定形式]

void	R_<Config_MTU0>_Stop (void);

[引数]

なし

[戻り値]

なし

R_<Config_MTU0>_Create_UserInit

ノーマルモード（MTU/TPU）に関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_MTU0>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_MTU0>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

r_<Config_MTU0>_tgi*mn*_interrupt

r_<Config_MTU0>_tgi*nm*_interrupt

インプットキャプチャ／コンペアマッチ割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_MTU0>_tgi*mn*_interrupt (void);

void	r_<Config_MTU0>_tgi*nm*_interrupt (void);

備考 *n* はチャネル番号を、*m* はタイマジェネラルレジスタ番号を意味します。

備考 リソースにより、API 関数名が異なります。

[引数]

なし

[戻り値]

なし

r_<Config_MTU0>_tciv*n*_interrupt

r_<Config_MTU0>_tci*n*v_interrupt

オーバフロー割り込みの発生に伴う処理を行います。

1

```
void        r_<Config_MTU0>_tciv n _interrupt ( void );
```

```
void        r_<Config_MTU0>_tci n v_interrupt ( void );
```

備考　*n* はチャネル番号を意味します。

備考　リソースにより、API 関数名が異なります。

[引数]

なし

[戻り値]

なし

使用例

ワンショットタイマとして使用する

main.c

```
#include "r_smc_entry.h"
void main(void)
{
    /* Start MTU channel 0 counter */
    R_Config_MTU0_Start();

    while (1U)
    {
        nop();
    }
}
```

Config_MTU0_user.c

```
static void r_<Config_MTU0_tgia0_interrupt(void)
{
    /* Start user code for r_<Config_MTU0_tgia0_interrupt. Do not edit comment generated here */
    /* Stop MTU channel 0 counter */
    R_Config_MTU0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

4.2.24 位相計数モードタイマ

以下に、コード生成ツールが位相計数モードタイマ（MTU/TPU）用として出力する API 関数の一覧を示します。

表 4.25 位相計数モードタイマ用 API 関数

API 関数名	機能概要
R_<Config_MTU1>_Create	位相計数モードタイマ（MTU/TPU）を制御するうえで必要となる初期化処理を行います。
R_<Config_MTU1>_Start	カウンタを開始します。
R_<Config_MTU1>_Stop	カウンタを終了します。
R_<Config_MTU1_MTU2>_MTU_Start	32 ビットカスケード動作のために MTU のすべてチャンネルのカウンタを同時に開始します。
R_<Config_MTU1_MTU2>_MTU_Stop	32 ビットカスケード動作のために MTU のすべてのチャンネルのカウンタを同時に終了します。
R_<Config_MTU1>_Create_UserInit	位相計数モードタイマ（MTU/TPU）に関するユーザ独自の初期化処理を行います。
r_<Config_MTU1>_tgimn_interrupt	インプットキャプチャ／コンペアマッチ割り込みの発生に伴う処理を行います。 （リソースにより、API 関数名が異なります。）
r_<Config_MTU1>_tginm_interrupt	
r_<Config_MTU1>_tcivn_interrupt	オーバフロー割り込みの発生に伴う処理を行います。 （リソースにより、API 関数名が異なります。）
r_<Config_MTU1>_tcinv_interrupt	
r_<Config_MTU1>_tciun_interrupt	アンダフロー割り込みの発生に伴う処理を行います。 （リソースにより、API 関数名が異なります。）
r_<Config_MTU1>_tcinu_interrupt	

R_<Config_MTU1>_Create

位相計数モードタイマ（MTU/TPU）を制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void    R_<Config_MTU1>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_MTU1>_Start

カウンタを開始します。

[指定形式]

```
void        R_<Config_MTU1>_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_MTU1>_Stop

カウンタを終了します。

[指定形式]

```
void        R_<Config_MTU1>_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_MTU1_MTU2>_MTU_Start

32 ビットカスケード動作で使用する MTU チャンネルのカウンタを同時に開始します。カスケード動作に関連するチャンネルは、カスケードモードの GUI から設定できます。

[指定形式]

Void	R_<Config_MTU1_MTU2>_MTU_Start (void);

[引数]

なし

[戻り値]

なし

R_<Config_MTU1_MTU2>_MTU_Stop

32 ビットカスケード動作で使用する MTU チャンネルのカウンタを同時に終了します。カスケード動作に関連するチャンネルは、カスケードモードの GUI から設定できます。

[指定形式]

```
void        R_<Config_MTU1_MTU2>_MTU_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_MTU1>_Create_UserInit

位相計数モードタイマ（MTU/TPU）に関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_MTU1>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_MTU1>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

r_<Config_MTU1>_tgi*mn*_interrupt

r_<Config_MTU1>_tgi*nm*_interrupt

インプットキャプチャ／コンペアマッチ割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_MTU1>_tgi*mn*_interrupt (void);

void	r_<Config_MTU1>_tgi*nm*_interrupt (void);

備考 *n* はチャネル番号を、*m* はタイマジェネラルレジスタ番号を意味します。

備考 リソースにより、API 関数名が異なります。

[引数]

なし

[戻り値]

なし

r_<Config_MTU1>_tciv*n*_interrupt

r_<Config_MTU1>_tci*n*v_interrupt

オーバフロー割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_MTU1>_tciv*n*_interrupt (void);

void	r_<Config_MTU1>_tci*n*v_interrupt (void);

備考 *n* はチャネル番号を意味します。

備考 リソースにより、API 関数名が異なります。

[引数]

なし

[戻り値]

なし

r_<Config_MTU1>_tciu*n*_interrupt

r_<Config_MTU1>_tci*n*u_interrupt

アンダフロー割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_MTU1>_tciu*n*_interrupt (void);

void	r_<Config_MTU1>_tci*n*u_interrupt (void);

備考　*n* はチャネル番号を意味します。

備考　リソースにより、API 関数名が異なります。

[引数]

なし

[戻り値]

なし

使用例

2 相の入力パルスにより、エンコーダ回転方向の確定を行う

main.c

```
#include "r_smc_entry.h"
void main(void)
{
    /* Start the MTU1 channel counter */
    R_Config_MTU1_Start();

    while (1U)
    {
        nop();
    }
}
```

Config_MTU1_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_mtu1_dir;
/* End user code. Do not edit comment generated here */

void R_Config_MTU1_Create_UserInit(void)
{
    /* Start user code for user init. Do not edit comment generated here */
    /* Clear state */
    g_mtu1_dir = 0U;
    /* End user code. Do not edit comment generated here */
}

static void r_Config_MTU1_tgia1_interrupt(void)
{
    /* Start user code for r_Config_MTU1_tgia1_interrupt. Do not edit comment generated here */
    /* Set CW state */
    g_mtu1_dir = 1U;
    /* End user code. Do not edit comment generated here */
}

static void r_Config_MTU1_tgib1_interrupt(void)
{
    /* Start user code for r_Config_MTU1_tgib1_interrupt. Do not edit comment generated here */
    /* Set CCW state */
    g_mtu1_dir = 2U;
    /* End user code. Do not edit comment generated here */
}
```

4.2.25 ポートアウトプットイネーブル

以下に、コード生成ツールがポートアウトプットイネーブル用として出力する API 関数の一覧を示します。

表 4.26 ポートアウトプットイネーブル用 API 関数

API 関数名	機能概要
R_<Config_POE>_Create	ポートアウトプットイネーブルを制御するうえで必要となる初期化処理を行います。
R_<Config_POE>_Start	割り込みを許可します。
R_<Config_POE>_Stop	割り込みを禁止します。
R_<Config_POE>_Set_HiZ_MTUn	MTU*n* 端子をハイインピーダンス状態にします。
R_<Config_POE>_Clear_HiZ_MTUn	MTU*n* 端子のハイインピーダンス状態を解除します。
R_<Config_POE>_Set_HiZ_GPTn	GPT*n* 端子をハイインピーダンス状態にします。
R_<Config_POE>_Clear_HiZ_GPTn	GPT*n* 端子のハイインピーダンス状態を解除します。
R_<Config_POE>_Create_UserInit	ポートアウトプットイネーブルに関するユーザ独自の初期化処理を行います。
r_<Config_POE>_oein_interrupt	アウトプットイネーブル割り込みの発生に伴う処理を行います。

R_<Config_POE>_Create

ポートアウトプットイネーブルを制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_<Config_POE>_Start

割り込みを許可します。

[指定形式]

void	R_<Config_POE>_Start (void);

[引数]

なし

[戻り値]

なし

R_<Config_POE>_Stop

割り込みを禁止します。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_<Config_POE>_Set_HiZ_MTU*n*

MTU*n* 端子をハイインピーダンス状態にします。

[指定形式]

void	R_<Config_POE>_Set_HiZ_MTU*n* (void);

備考 *n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_<Config_POE>_Clear_HiZ_MTU*n*

MTU*n* 端子のハイインピーダンス状態を解除します。

[指定形式]

void	R_<Config_POE>_Clear_HiZ_MTU*n* (void);

備考 *n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_<Config_POE>_Set_HiZ_GPT*n*

GPT*n* 端子をハイインピーダンス状態にします。

[指定形式]

void	R_<Config_POE>_Set_HiZ_GPT*n* (void);

備考 *n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_<Config_POE>_Clear_HiZ_GPT*n*

GPT*n* 端子のハイインピーダンス状態を解除します。

[指定形式]

void	R_<Config_POE>_Clear_HiZ_GPT*n* (void);

備考 *n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_<Config_POE>_Create_UserInit

ポートアウトプットイネーブルに関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_POE>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_POE>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

r_<Config_POE>_oei*n*_interrupt

アウトプットイネーブル割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_POE>_oei*n*_interrupt (void);

備考 *n* はアウトプットイネーブル割り込み要因番号を意味します。

[引数]

なし

[戻り値]

なし

使用例

アウトプットイネーブル割り込み時に MTU0 端子もハイインピーダンスに設定する

main.c

```
#include "r_smc_entry.h"
void main(void)
{
    /* Start the POE module */
    R_Config_POE_Start();

    while (1U)
    {
        nop();
    }
}
```

Config_POE_user.c

```
void r_Config_POE_oei1_interrupt(void)
{
    /* Start user code for r_Config_POE_oei1_interrupt. Do not edit comment generated here */
    /* Stop the POE module */
    R_Config_POE_Stop();

    /* Place MTU0 pins in high-impedance */
    R_Config_POE_Set_HiZ_MTU0();
    /* End user code. Do not edit comment generated here */
}
```

4.2.26 ポート

以下に、コード生成ツールがポート用として出力する API 関数の一覧を示します。

表 4.27 ポート用 API 関数

API 関数名	機能概要
R_<Config_PORT>_Create	ポートを制御するうえで必要となる初期化処理を行います。
R_<Config_PORT>_Create_UserInit	ポートに関するユーザ独自の初期化処理を行います。

R_<Config_PORT>_Create

ポートを制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_<Config_PORT>_Create_UserInit

ポートに関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_PORT>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_PORT>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

使用例

なし

4.2.27 プログラマブルパルスジェネレータ

以下に、コード生成ツールがプログラマブルパルスジェネレータ用として出力する API 関数の一覧を示します。

表 4.28 プログラマブルパルスジェネレータ用 API 関数

API 関数名	機能概要
R_<Config_PPG0>_Create	プログラマブルパルスジェネレータを制御するうえで必要となる初期化処理を行います。
R_<Config_PPG0>_Create_UserInit	プログラマブルパルスジェネレータに関するユーザ独自の初期化処理を行います。

R_<Config_PPG0>_Create

プログラマブルパルスジェネレータを制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void    R_<Config_PPG0>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_PPG0>_Create_UserInit

プログラマブルパルスジェネレータに関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_PPG0>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_PPG0>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

使用例

なし

4.2.28 PWM モードタイマ

以下に、コード生成ツールが PWM モードタイマ（MTU/TPU）用として出力する API 関数の一覧を示します。

表 4.29 PWM モードタイマ用 API 関数

API 関数名	機能概要
R_<Config_MTU0>_Create	PWM モードタイマ（MTU/TPU）を制御するうえで必要となる初期化処理を行います。
R_<Config_MTU0>_Start	カウンタを開始します。
R_<Config_MTU0>_Stop	カウンタを終了します。
R_<Config_MTU0>_Create_UserInit	PWM モードタイマ（MTU/TPU）に関するユーザ独自の初期化処理を行います。
r_<Config_MTU0>_tgimn_interrupt	コンペアマッチ割り込みの発生に伴う処理を行います。 （リソースにより、API 関数名が異なります。）
r_<Config_MTU0>_tginm_interrupt	
r_<Config_MTU0>_tcivn_interrupt	オーバフロー割り込みの発生に伴う処理を行います。 （リソースにより、API 関数名が異なります。）
r_<Config_MTU0>_tcinv_interrupt	

R_<Config_MTU0>_Create

PWM モードタイマ（MTU/TPU）を制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void    R_<Config_MTU0>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_MTU0>_Start

カウンタを開始します。

[指定形式]

void	R_<Config_MTU0>_Start (void);

[引数]

なし

[戻り値]

なし

R_<Config_MTU0>_Stop

カウンタを終了します。

[指定形式]

```
void        R_<Config_MTU0>_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_MTU0>_Create_UserInit

PWM モードタイマ（MTU/TPU）に関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_MTU0>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_MTU0>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

r_<Config_MTU0>_tgi*mn*_interrupt

r_<Config_MTU0>_tgi*nm*_interrupt

コンペアマッチ割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_MTU0>_tgi*mn*_interrupt (void);

void	r_<Config_MTU0>_tgi*nm*_interrupt (void);

備考 1　*n* はチャネル番号を、*m* はタイマジェネラルレジスタ番号を意味します。
備考 2　リソースにより、API 関数名が異なります。

[引数]

なし

[戻り値]

なし

r_<Config_MTU0>_tciv*n*_interrupt

r_<Config_MTU0>_tci*n*v_interrupt

オーバフロー割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_MTU0>_tciv*n*_interrupt (void);

void	r_<Config_MTU0>_tci*n*v_interrupt (void);

備考 1　n はチャネル番号を意味します。
備考 2　リソースにより、API 関数名が異なります。

[引数]

なし

[戻り値]

なし

使用例

PWM 出力を 10 回発生させる

main.c

```
#include "r_smc_entry.h"
void main(void)
{
    /* Start MTU channel 0 counter */
    R_Config_MTU0_Start();

    while (1U)
    {
        nop();
    }
}
```

Config_MTU0_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_mtu0_cnt;
/* End user code. Do not edit comment generated here */
void R_Config_MTU0_Create_UserInit(void)
{
    /* Start user code for user init. Do not edit comment generated here */
    /* Reset the countor */
    g_mtu0_cnt = 0U;
    /* End user code. Do not edit comment generated here */
}

static void r_Config_MTU0_tgia0_interrupt(void)
{
    /* Start user code for r_Config_MTU0_tgia0_interrupt. Do not edit comment generated here */
    if ((++g_mtu0_cnt) > 9U)
    {
        /* Stop MTU channel 0 counter */
        R_Config_MTU0_Stop();
    }
    /* End user code. Do not edit comment generated here */
}
```

4.2.29 リアルタイムクロック

以下に、コード生成ツールがリアルタイムクロック用として出力する API 関数の一覧を示します。

表 4.30 リアルタイムクロック用 API 関数

API 関数名	機能概要
R_<Config_RTC>_Create	リアルタイムクロックを制御するうえで必要となる初期化処理を行います。
R_<Config_RTC>_Start	カウンタを開始します。
R_<Config_RTC>_Stop	カウンタを終了します。
R_<Config_RTC>_Restart	カウンタを終了し、カウンタを初期化したのち、カウンタを再開します。
R_<Config_RTC>_Restart_BinaryCounter	カウンタを終了し、カウンタを再開します。
R_<Config_RTC>_Set_CalendarCounterValue	カレンダ値を設定します。
R_<Config_RTC>_Get_CalendarCounterValue	カレンダ値を獲得します。
R_<Config_RTC>_Get_CalendarTimeCaptureValuen	キャプチャしたカレンダ値を獲得します。
R_<Config_RTC>_Set_BinaryCounterValue	バイナリカウント値を設定します。
R_<Config_RTC>_Get_BinaryCounterValue	バイナリカウント値を獲得します。
R_<Config_RTC>_Get_BinaryTimeCaptureValuen	キャプチャしたバイナリカウント値を獲得します。
R_<Config_RTC>_Set_RTCOUTOn	RTCOUT への出力周期を設定すると伴に、RTCOUT 出力を開始します。
R_<Config_RTC>_Set_RTCOUTOff	RTCOUT 出力を終了します。
R_<Config_RTC>_Set_CalendarAlarm	アラーム割り込みの発生条件を設定するとともに、アラーム割り込みの検出を許可します。（カレンダカウントモード）
R_<Config_RTC>_Set_BinaryAlarm	アラーム割り込みの発生条件を設定するとともに、アラーム割り込みの検出を許可します。（バイナリカウントモード）
R_<Config_RTC>_Set_ConstPeriodInterruptOn	周期割り込みの発生周期を設定すると伴に、周期割り込みの検出を許可します。
R_<Config_RTC>_Set_ConstPeriodInterruptOff	周期割り込みの検出を禁止します。
R_<Config_RTC>_Set_CarryInterruptOn	桁上げ割り込みの検出を許可します。
R_<Config_RTC>_Set_CarryInterruptOff	桁上げ割り込みの検出を禁止します。
R_<Config_RTC>_Enable_Alarm_Interruptf	アラーム割り込みを許可します。
R_<Config_RTC>_Disable_Alarm_Interrupt	アラーム割り込みを禁止します。
R_<Config_RTC>_Create_UserInit	リアルタイムクロックに関するユーザ独自の初期化処理を行います。
r_<Config_RTC>_alm_interrupt	アラーム割り込みの発生に伴う処理を行います。
r_<Config_RTC>_prd_interrupt	周期割り込みの発生に伴う処理を行います。
r_<Config_RTC>_cup_interrupt	桁上げ割り込みの発生に伴う処理を行います。

R_<Config_RTC>_Create

リアルタイムクロックを制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_<Config_RTC>_Start

カウンタを開始します。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_<Config_RTC>_Stop

カウンタを終了します。

[指定形式]

void	R_<Config_RTC>_Stop (void);

[引数]

なし

[戻り値]

なし

R_<Config_RTC>_Restart

カウンタを終了し、カウンタを初期化したのち、カウンタを再開します。

[指定形式]

void	R_<Config_RTC>_Restart (rtc_calendarcounter_value_t *counter_write_val*);

[引数]

I/O	引数	説明
I	rtc_calendarcounter_value_t *counter_write_val*;	初期値（年， 月， 日， 曜日， 時， 分， 秒）

備考 以下に、初期値 rtc_calendarcounter_value_t の構成を示します。

```
typedef struct {
    uint8_t rseccnt;        /*  秒  */
    uint8_t rmincnt;        /*  分  */
    uint8_t rhrcnt;         /*  時  */
    uint8_t rdaycnt;        /*  日  */
    uint8_t rwkcnt;         /*  曜日（ 0 ：日曜日，  6 ：土曜日）  */
    uint8_t rmoncnt;        /*  月  */
    uint16_t ryrcnt;        /*  年  */
} rtc_calendarcounter_value_t;
```

[戻り値]

なし

R_<Config_RTC>_Restart_BinaryCounter

カウンタを終了し、カウンタを再開します。

[指定形式]

```
void        R_<Config_RTC>_Restart_BinaryCounter ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_RTC>_Set_CalendarCounterValue

カレンダ値を設定します。

[指定形式]

```
void    R_<Config_RTC>_Set_CalendarCounterValue ( rtc_calendarcounter_value_t counter_write_val );
```

[引数]

I/O	引数	説明
I	rtc_calendarcounter_value_t *counter_write_val*;	カレンダ値（年，月，日，曜日，時，分，秒）

備考　以下に、カレンダ値 rtc_calendarcounter_value_t の構成を示します。

```
typedef struct {
    uint8_t rseccnt;      /*  秒  */
    uint8_t rmincnt;      /*  分  */
    uint8_t rhrcnt;       /*  時  */
    uint8_t rdaycnt;      /*  日  */
    uint8_t rwkcnt;       /*  曜日（ 0 ：日曜日，  6 ：土曜日）  */
    uint8_t rmoncnt;      /*  月  */
    uint16_t ryrcnt;      /*  年  */
} rtc_calendarcounter_value_t;
```

[戻り値]

なし

R_<Config_RTC>_Get_CalendarCounterValue

カレンダ値を獲得します。

[指定形式]

```
void R_<Config_RTC>_Get_CalendarCounterValue ( rtc_calendarcounter_value_t * const counter_read_val );
```

[引数]

I/O	引数	説明
O	rtc_calendarcounter_value_t * const *counter_read_val*;	獲得したカレンダ値（年，月，日，曜日，時，分，秒）を格納する領域へのポインタ

備考 以下に、カレンダ値 rtc_calendarcounter_value_t の構成を示します。

```
typedef struct {
    uint8_t rseccnt;    /*  秒  */
    uint8_t rmincnt;    /*  分  */
    uint8_t rhrcnt;     /*  時  */
    uint8_t rdaycnt;    /*  日  */
    uint8_t rwkcnt;     /*  曜日（ 0 ：日曜日，  6 ：土曜日）  */
    uint8_t rmoncnt;    /*  月  */
    uint16_t ryrcnt;    /*  年  */
} rtc_calendarcounter_value_t;
```

[戻り値]

なし

R_<Config_RTC>_Get_CalendarTimeCaptureValue*n*

キャプチャしたカレンダ値を獲得します。

[指定形式]

```
void    R_<Config_RTC>_Get_CalendarTimeCaptureValuen ( rtc_calendarcounter_value_t * const
counter_read_val );
```

備考　　*n* は、チャネル番号を意味します。

[引数]

I/O	引数	説明
O	rtc_calendarcounter_value_t * const *counter_read_val*;	獲得したカレンダ値（年，月，日，曜日，時，分，秒）を格納する領域へのポインタ

備考　　以下に、カレンダ rtc_calendarcounter_value_t の構成を示します。

```
typedef struct {
    uint8_t rseccnt;        /*  秒  */
    uint8_t rmincnt;        /*  分  */
    uint8_t rhrcnt;         /*  時  */
    uint8_t rdaycnt;        /*  日  */
    uint8_t rwkcnt;         /*  曜日（ 0 ：日曜日，  6 ：土曜日）  */
    uint8_t rmoncnt;        /*  月  */
    uint16_t ryrcnt;        /*  年  */
} rtc_calendarcounter_value_t;
```

[戻り値]

なし

R_<Config_RTC>_Set_BinaryCounterValue

バイナリカウント値を設定します。

[指定形式]

void	R_<Config_RTC>_Set_BinaryCounterValue (uint32_t *counter_write_val*);

[引数]

I/O	引数	説明
I	uint32_t *counter_write_val*;	バイナリカウント値

[戻り値]

なし

R_<Config_RTC>_Get_BinaryCounterValue

バイナリカウント値を獲得します。

[指定形式]

void	R_<Config_RTC>_Get_BinaryCounterValue (uint32_t * const *counter_read_val*);

[引数]

I/O	引数	説明
O	uint32_t * const *counter_read_val*;	獲得したバイナリカウント値を格納する領域へのポインタ

[戻り値]

なし

R_<Config_RTC>_Get_BinaryTimeCaptureValue*n*

キャプチャしたバイナリカウント値を獲得します。

[指定形式]

void	R_<Config_RTC>_Get_BinaryTimeCaptureValue*n* (uint32_t * const *counter_read_val*);

備考 *n* は、チャネル番号を意味します。

[引数]

I/O	引数	説明
O	uint32_t * const *counter_read_val*;	獲得したバイナリカウント値を格納する領域へのポインタ

[戻り値]

なし

R_<Config_RTC>_Set_RTCOUTOn

RTCOUT への出力周期を設定すると伴に、RTCOUT 出力を開始します。

[指定形式]

void R_<Config_RTC>_Set_RTCOUTOn (rtc_rtcout_period_t *rtcout_freq*);

[引数]

I/O	引数	説明
I	rtc_rtcout_period_t *rtcout_freq*;	RTCOUT への出力周期 RTCOUT_1HZ : 1Hz RTCOUT_64HZ : 64Hz

[戻り値]

なし

R_<Config_RTC>_Set_RTCOUTOff

RTCOUT 出力を終了します。

[指定形式]

void	R_<Config_RTC>_Set_RTCOUTOff (void);

[引数]

なし

[戻り値]

なし

R_<Config_RTC>_Set_CalendarAlarm

アラーム割り込みの発生条件を設定すると伴に、アラーム割り込みの検出を許可します。（カレンダカウントモード）

[指定形式]

```
void      R_<Config_RTC>_Set_CalendarAlarm ( rtc_calendar_alarm_enable_t alarm_enable、
 rtc_calendar_alarm_value_t alarm_val );
```

[引数]

I/O	引数	説明
I	rtc_calendar_alarm_enable_t *alarm_enable*;	比較フラグ（年，月，日，曜日，時，分，秒）
I	rtc_calendar_alarm_value_t *alarm_val*;	カレンダ値（年，月，日，曜日，時，分，秒）

備考 1.　以下に、比較フラグ rtc_calendar_alarm_enable_t の構成を示します。

```
typedef struct {
    uint8_t sec_enb;  /*  秒（ 0x0 ：比較を行わない、 0x80 ：比較を行う）*/
    uint8_t min_enb;  /*  分（ 0x0 ：比較を行わない、 0x80 ：比較を行う）*/
    uint8_t hr_enb;   /*  時（ 0x0 ：比較を行わない、 0x80 ：比較を行う）*/
    uint8_t day_enb;  /*  日（ 0x0 ：比較を行わない、 0x80 ：比較を行う）*/
    uint8_t wk_enb;   /*  曜日（ 0x0 ：比較を行わない、 0x80 ：比較を行う） */
    uint8_t mon_enb;  /*  月（ 0x0 ：比較を行わない、 0x80 ：比較を行う）*/
    uint8_t yr_enb;   /*  年（ 0x0 ：比較を行わない、 0x80 ：比較を行う）*/
} rtc_calendar_alarm_enable_t;
```

備考 2.　以下に、カレンダ値 rtc_calendar_alarm_value_t の構成を示します。

```
typedef struct {
    uint8_t rsecar;       /*  秒  */
    uint8_t rminar;       /*  分  */
    uint8_t rhrar;        /*  時  */
    uint8_t rdayar;       /*  日  */
    uint8_t rwkar;        /*  曜日（ 0 ：日曜日，  6 ：土曜日）  */
    uint8_t rmonar;       /*  月  */
    uint16_t ryrar;       /*  年  */
} rtc_calendar_alarm_value_t;
```

[戻り値]

なし

R_<Config_RTC>_Set_BinaryAlarm

アラーム割り込みの発生条件を設定すると伴に、アラーム割り込みの検出を許可します。（バイナリカウントモード）

[指定形式]

void	R_<Config_RTC>_Set_BinaryAlarm (uint32_t *alarm_enable*、 uint32_t *alarm_val*);

[引数]

I/O	引数	説明
I	uint32_t *alarm_enable*;	比較フラグ 0x0 ：比較を行わない 0x1 ：比較を行う
I	uint32_t *alarm_val*;	バイナリカウント値

[戻り値]

なし

R_<Config_RTC>_Set_ConstPeriodInterruptOn

周期割り込みの発生周期を設定すると伴に、周期割り込みの検出を許可します。

[指定形式]

void	R_<Config_RTC>_Set_ConstPeriodInterruptOn (rtc_int_period_t *period*);

[引数]

I/O	引数	説明
I	rtc_int_period_t *period*;	周期割り込みの発生周期 PES_2_SEC : 2 秒 PES_1_SEC : 1 秒 PES_1_2_SEC : 1/2 秒 PES_1_4_SEC : 1/4 秒 PES_1_8_SEC : 1/8 秒 PES_1_16_SEC : 1/16 秒 PES_1_32_SEC : 1/32 秒 PES_1_64_SEC : 1/64 秒 PES_1_128_SEC : 1/128 秒 PES_1_256_SEC : 1/256 秒

[戻り値]

なし

R_<Config_RTC>_Set_ConstPeriodInterruptOff

周期割り込みの検出を禁止します。

[指定形式]

void	R_<Config_RTC>_Set_ConstPeriodInterruptOff (void);

[引数]

なし

[戻り値]

なし

R_<Config_RTC>_Set_CarryInterruptOn

桁上げ割り込みの検出を許可します。

[指定形式]

void	R_<Config_RTC>_Set_CarryInterruptOn (void);

[引数]

なし

[戻り値]

なし

R_<Config_RTC>_Set_CarryInterruptOff

桁上げ割り込みの検出を禁止します。

[指定形式]

void	R_<Config_RTC>_Set_CarryInterruptOff (void);

[引数]

なし

[戻り値]

なし

R_<Config_RTC>_Enable_Alarm_Interruptf

アラーム割り込みを許可します。

[指定形式]

```
void        R_<Config_RTC>_Enable_Alarm_Interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_RTC>_Disable_Alarm_Interrupt

アラーム割り込みを禁止します。

[指定形式]

void	R_<Config_RTC>_Disable_Alarm_Interrupt (void);

[引数]

なし

[戻り値]

なし

R_<Config_RTC>_Create_UserInit

リアルタイムクロックに関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_RTC>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_RTC>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

r_<Config_RTC>_alm_interrupt

アラーム割り込みの発生に伴う処理を行います。

備考　本 API 関数は、R_<Config_RTC>_Set_CalendarAlarm で指定された条件を満足した場合に発生するアラーム割り込みに対応した割り込み処理として呼び出されます。

[指定形式]

void	r_<Config_RTC>_alm_interrupt (void);

[引数]

なし

[戻り値]

なし

r_<Config_RTC>_prd_interrupt

周期割り込みの発生に伴う処理を行います。

備考 本 API 関数は、R_<Config_RTC>_Set_ConstPeriodInterruptOn で指定された周期 *period* が経過した場合に発生する周期割り込みに対応した割り込み処理として呼び出されます。

[指定形式]

void	r_<Config_RTC>_prd_interrupt (void);

[引数]

なし

[戻り値]

なし

r_<Config_RTC>_cup_interrupt

桁上げ割り込みの発生に伴う処理を行います。

備考　本 API 関数は、秒カウンタ（RSECCNT）／バイナリカウンタ 0（BCNT0）の桁上げを行った場合、または 64Hz カウンタ（R64CNT）の読み出しと 64Hz カウンタ（R64CNT）の桁上げが重複した場合に発生する桁上げ割り込みに対応した割り込み処理として呼び出されます。

[指定形式]

```
void    r_<Config_RTC>_cup_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

使用例

アラーム割り込みで擬似的に閏秒の補正を行う（特定日の日付変更直前の 59 秒時点で 58 秒に戻す）

main.c

```
#include "r_smc_entry.h"
void main(void)
{
    /* Start RTC counter */
    R_Config_RTC_Start();

    while (1U)
    {
        nop();
    }
}
```

Config_RTC_user.c

```
/* Start user code for include. Do not edit comment generated here */
volatile rtc_calendarcounter_value_t counter_val;
/* End user code. Do not edit comment generated here */
static void r_Config_RTC_alm_interrupt(void)
{
    /* Start user code for r_Config_RTC_alm_interrupt. Do not edit comment generated here */
    /* Disable ALM interrupt */
    IEN(RTC、ALM) = 0U;

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

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

    /* Set RTC calendar counter value */
    R_Config_RTC_Set_CalendarCounterValue(counter_val);
    /* End user code. Do not edit comment generated here */
}
```

4.2.30 リモコン信号受信機能

以下に、コード生成ツールがリモコン信号受信機能用として出力する API 関数の一覧を示します。

表 4.31 リモコン信号受信機能用 API 関数

API 関数名	機能概要
R_<Config_REMC0>_Create	リモコン信号受信機能を制御するうえで必要となる初期化処理を行います。
R_<Config_REMC0>_Start	リモコン信号受信機能の動作を開始します。
R_<Config_REMC0>_Stop	リモコン信号受信機能の動作を停止します。
R_<Config_REMC0>_Read	受信データの読み出しを開始します。
R_<Config_REMC0>_Create_UserInit	リモコン信号受信機能に関するユーザ独自の初期化処理を行います。
r_<Config_REMC0>_remcin_interrupt	REMC 割り込みに伴う処理を行います。
r_<Config_REMC0>_callback_comparematch	コンペアー致割り込みに伴う処理を行います。
r_<Config_REMC0>_callback_receiveerror	受信エラー割り込みに伴う処理を行います。
r_<Config_REMC0>_callback_receiveend	データ受信完了割り込みに伴う処理を行います。
r_<Config_REMC0>_callback_bufferfull	受信バッファフル割り込みに伴う処理を行います。
r_<Config_REMC0>_callback_header	ヘッダパターン一致割り込みに伴う処理を行います。
r_<Config_REMC0>_callback_data0	データ“0”パターンの一致割り込みに伴う処理を行います。
r_<Config_REMC0>_callback_data1	データ“1”パターンの一致割り込みに伴う処理を行います。
r_<Config_REMC0>_callback_specialdata	特殊データパターン一致割り込みに伴う処理を行います。

R_<Config_REMC0>_Create

リモコン信号受信機能を制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void    R_<Config_REMC0>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_REMC0>_Start

リモコン信号受信機能の動作を開始します。

[指定形式]

```
void        R_<Config_REMC0>_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_REMC0>_Stop

リモコン信号受信機能の動作を停止します。

[指定形式]

```
void        R_<Config_REMC0>_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_REMC0>_Read

受信データの読み出しを開始します。

備考　本 API 関数は、データ受信完了時に REMC 割り込みルーチン内で読み出す受信データの格納先を設定します。

[指定形式]

MD_STATUS	R_<Config_REMC0>_Read (uint8_t * const *rx_buf*, uint8_t *rx_num*);

[引数]

I/O	引数	説明
O	uint8_t * const *rx_buf*	受信したデータを格納するバッファへのポインタ
I	uint8_t *rx_num*	受信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR1	引数 *rx_num* の指定が不正

R_<Config_REMC0>_Create_UserInit

リモコン信号受信機能に関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_REMC0>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void    R_<Config_REMC0>_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_REMC0>_remci*n*_interrupt

REMC 割り込みに伴う処理を行います。

[指定形式]

void	r_<Config_REMC0>_remci*n*_interrupt (void);

備考 *n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

r_<Config_REMC0>_callback_comparematch

コンペアー致割り込みに伴う処理を行います。

備考 API 関数は、REMC 割り込みに対応した割り込み処理 r_<Config_REMC0>_remcin_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void r_<Config_REMC0>_callback_comparematch ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_REMC0>_callback_receiveerror

受信エラー割り込みに伴う処理を行います。

備考 API 関数は、REMC 割り込みに対応した割り込み処理 r_<Config_REMC0>_remcin_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	r_<Config_REMC0>_callback_receiveerror (void);

[引数]

なし

[戻り値]

なし

r_<Config_REMC0>_callback_receiveend

データ受信完了割り込みに伴う処理を行います。

備考 API 関数は、REMC 割り込みに対応した割り込み処理 r_<Config_REMC0>_remcin_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void        r_<Config_REMC0>_callback_receiveend ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_REMC0>_callback_bufferfull

受信バッファフル割り込みに伴う処理を行います。

備考 API 関数は、REMC 割り込みに対応した割り込み処理 r_<Config_REMC0>_remcin_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void        r_<Config_REMC0>_callback_bufferfull ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_REMC0>_callback_header

ヘッダパターン一致割り込みに伴う処理を行います。

備考 API 関数は、REMC 割り込みに対応した割り込み処理 r_<Config_REMC0>_remcin_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void        r_<Config_REMC0>_callback_header ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_REMC0>_callback_data0

データ“0”パターンの一致割り込みに伴う処理を行います。

備考 API 関数は、REMC 割り込みに対応した割り込み処理 r_<Config_REMC0>_remcin_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void r_<Config_REMC0>_callback_data0 ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_REMC0>_callback_data1

データ“1”パターンの一致割り込みに伴う処理を行います。

備考 API 関数は、REMC 割り込みに対応した割り込み処理 r_<Config_REMC0>_remcin_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void r_<Config_REMC0>_callback_data1 ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_REMC0>_callback_specialdata

特殊データパターン一致割り込みに伴う処理を行います。

備考 API 関数は、REMC 割り込みに対応した割り込み処理 r_<Config_REMC0>_remcin_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	r_<Config_REMC0>_callback_specialdata (void);

[引数]

なし

[戻り値]

なし

使用例

データ受信完了でリモコン受信を停止する

main.c

```
#include "r_smc_entry.h"
extern volatile uint8_t g_remc0_rx_buf[8];
void main(void)
{
    /* Start the REMC0 operation */
    R_Config_REMC0_Start();

    /* Read data from receive data buffer */
    R_Config_REMC0_Read((uint8_t *)g_remc0_rx_buf, 8U);

    while (1U)
    {
        nop();
    }
}
```

Config_REMC0_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_remc0_rx_buf[8];
/* End user code. Do not edit comment generated here */
static void r_Config_REMC0_callback_receiveend(void)
{
    /* Start user code for r_Config_REMC0_callback_receiveend. Do not edit comment generated
here */
    /* Stop the REMC0 operation */
    R_Config_REMC0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

4.2.31 SCI/SCIF 調歩同期式モード

以下に、コード生成ツールが SCI/SCIF 調歩同期式モード用として出力する API 関数の一覧を示します。

表 4.32 SCI/SCIF/RSCI 調歩同期式モード用 API 関数

API 関数名	機能概要
R_<Config_SCI0>_Create	SCI/SCIF 調歩同期式モードを制御するうえで必要となる初期化処理を行います。
R_<Config_SCI0>_Start	通信を開始します。
R_<Config_SCI0>_Stop	通信を終了します。
R_<Config_SCI0>_Serial_Send	送信を開始します。（調歩同期式モード）
R_<Config_SCI0>_Serial_Receive	受信を開始します。（調歩同期式モード）
R_<Config_SCI0>_Serial_Multiprocessor_Send	【 SCI/RSCI 】送信を開始します。（マルチプロセッサ通信機能）
R_<Config_SCI0>_Serial_Multiprocessor_Receive	【 SCI/RSCI 】受信を開始します。（マルチプロセッサ通信機能）
R_<Config_SCI0>_Create_UserInit	SCI/SCIF 調歩同期式モードに関するユーザ独自の初期化処理を行います。
r_<Config_SCI0>_transmitend_interrupt	【 SCI/RSCI 】送信終了割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_transmit_interrupt	【 SCI/RSCI 】送信データエンプティ割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_receive_interrupt	【 SCI/RSCI 】受信データフル割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_receiveerror_interrupt	【 SCI/RSCI 】受信エラー割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_teif_interrupt	【 SCIF 】送信完了割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_txif_interrupt	【 SCIF 】送信 FIFO データエンプティ割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_rxif_interrupt	【 SCIF 】受信 FIFO データフル割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_drif_interrupt	【 SCIF 】受信データレディ割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_erif_interrupt	【 SCIF 】受信エラー割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_brif_interrupt	【 SCIF 】ブレーク割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_callback_transmitend	【 SCI/RSCI 】送信終了／送信データエンプティ割り込みの発生に伴う処理を行います。 【 SCIF 】送信完了／送信 FIFO データエンプティ割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_callback_receiveend	【 SCI/RSCI 】受信データフル割り込みの発生に伴う処理を行います。 【 SCIF 】受信 FIFO データフル割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_callback_receiveerror	【 SCI/RSCI 】受信エラー割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_callback_error	【 SCIF 】受信エラー／ブレーク割り込みの発生に伴う処理を行います。

R_<Config_SCI0>_Create

SCI/SCIF 調歩同期式モードを制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void    R_<Config_SCI0>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_SCI0>_Start

通信を開始します。

[指定形式]

```
void        R_<Config_SCI0>_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_SCI0>_Stop

通信を終了します。

[指定形式]

```
void        R_<Config_SCI0>_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_SCI0>_Serial_Send

送信を開始します。(調歩同期式モード)

備考 1. 本 API 関数では、引数 *tx_buf* で指定されたバッファから 1 バイト単位の送信を引数 *tx_num* で指定された回数だけ繰り返し行います。

備考 2. 送信を行う際には、本 API 関数の呼び出し以前に R_<Config_SCI0>_Start を呼び出す必要があります。

[指定形式]

MD_STATUS	R_<Config_SCI0>_Serial_Send (uint8_t * const *tx_buf*, uint16_t *tx_num*);

[引数]

I/O	引数	説明
I	uint8_t * const *tx_buf*;	送信するデータを格納したバッファへのポインタ
I	uint16_t *tx_num*;	送信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数 *tx_num* の指定が不正

R_<Config_SCI0>_Serial_Receive

受信を開始します。(調歩同期式モード)

備考 1. 本 API 関数では、 1 バイト単位の受信を引数 *rx_num* で指定された回数だけ繰り返し行い、引数 *rx_buf* で指定されたバッファに格納します。

備考 2. 受信を行う際には、本 API 関数の呼び出し以前に R_<Config_SCI0>_Start を呼び出す必要があります。

[指定形式]

```
MD_STATUS     R_<Config_SCI0>_Serial_Receive ( uint8_t * const rx_buf, uint16_t rx_num );
```

[引数]

I/O	引数	説明
O	uint8_t * const *rx_buf*;	受信したデータを格納するバッファへのポインタ
I	uint16_t *rx_num*;	受信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数 *rx_num* の指定が不正

R_<Config_SCI0>_Serial_Multiprocessor_Send

【 SCI/RSCI 】送信を開始します。（マルチプロセッサ通信機能）

備考 1. マルチプロセッサ通信機能の送信は、受信局を指定する ID 送信サイクルと指定された受信局に対するデータ送信サイクルで構成されます。

備考 2. 本 API 関数では、ID 送信サイクルの処理として、引数 *id_buf* で指定されたバッファから 1 バイト単位の送信を引数 *id_num* で指定された回数だけ繰り返し行います。

備考 3. 本 API 関数では、データ送信サイクルの処理として、引数 *tx_buf* で指定されたバッファから 1 バイト単位の送信を引数 *tx_num* で指定された回数だけ繰り返し行います。

備考 4. 送信を行う際には、本 API 関数の呼び出し以前に R_<Config_SCI0>_Start を呼び出す必要があります。

[指定形式]

MD_STATUS R_<Config_SCI0>_Serial_Multiprocessor_Send (uint8_t * const *id_buf*, uint16_t *id_num*, uint8_t * const *tx_buf*, uint16_t *tx_num*);

[引数]

I/O	引数	説明
I	uint8_t * const *id_buf*;	送信する ID を格納したバッファへのポインタ
I	uint16_t *id_num*;	送信する ID の総数
I	uint8_t * const *tx_buf*;	送信するデータを格納したバッファへのポインタ
I	uint16_t *tx_num*;	送信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数 *tx_num* の指定が不正

R_<Config_SCI0>_Serial_Multiprocessor_Receive

【 SCI/RSCI 】受信を開始します。（マルチプロセッサ通信機能）

備考 1. マルチプロセッサ通信機能の受信は、受信した ID が自局の ID と一致した場合に続いて送信される通信データを受信します。

備考 2. 本 API 関数では、 1 バイト単位の受信を引数 *rx_num* で指定された回数だけ繰り返し行い、引数 *rx_buf* で指定されたバッファに格納します。

備考 3. 受信を行う際には、本 API 関数の呼び出し以前に R_<Config_SCI0>_Start を呼び出す必要があります。

[指定形式]

```
MD_STATUS R_<Config_SCI0>_Serial_Multiprocessor_Receive ( uint8_t * const rx_buf,
uint16_t rx_num );
```

[引数]

I/O	引数	説明
O	uint8_t * const *rx_buf*;	受信したデータを格納するバッファへのポインタ
I	uint16_t *rx_num*;	受信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数 *rx_num* の指定が不正

R_<Config_SCI0>_Create_UserInit

SCI/SCIF 調歩同期式モードに関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_SCI0>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_SCI0>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_transmitend_interrupt

【 SCI/RSCI 】送信終了割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_SCI0>_transmitend_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_transmit_interrupt

【 SCI/RSCI 】送信データエンプティ割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_SCI0>_transmit_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_receive_interrupt

【 SCI/RSCI 】受信データフル割り込みの発生に伴う処理を行います。

このハンドラ内で受信割込み要求禁止(SCI0.SCR.BIT.RIE=0 / RSCI10.SCR0.BIT.RIE=0) を行っています。これが問題となる場合は、お客様の判断でソースコードの編集を行ってください。

[指定形式]

```
void        r_<Config_SCI0>_receive_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_receiveerror_interrupt

【 SCI/RSCI 】受信エラー割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_SCI0>_receiveerror_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_teif_interrupt

【 SCIF 】送信完了割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_SCI0>_teif_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_txif_interrupt

【 SCIF 】送信 FIFO データエンプティ割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_SCI0>_txif_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_rxif_interrupt

【 SCIF 】受信 FIFO データフル割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_SCI0>_rxif_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_drif_interrupt

【 SCIF 】受信データレディ割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_SCI0>_drif_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_erif_interrupt

【 SCIF 】受信エラー割り込みの発生に伴う処理を行います。

備考　本 API 関数は、フレーミングエラーまたはパリティエラーによる割り込みに対応した割り込み処理として呼び出されます。

[指定形式]

```
void    r_<Config_SCI0>_erif_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_brif_interrupt

【 SCIF 】ブレーク割り込みの発生に伴う処理を行います。

備考　本 API 関数は、ブレークまたはオーバランによる割り込みに対応した割り込み処理として呼び出されます。

[指定形式]

```
void    r_<Config_SCI0>_brif_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_callback_transmitend

【 SCI/RSCI 】送信終了／送信データエンプティ割り込みの発生に伴う処理を行います。

備考 1. 本 API 関数は、r_<Config_SCI0>_transmitend_interrupt および r_<Config_SCI0>_transmit_interrupt のコールバック・ルーチンとして呼び出されます。

【 SCIF 】送信完了／送信 FIFO データエンプティ割り込みの発生に伴う処理を行います。

備考 2. 本 API 関数は、r_<Config_SCI0>_teif_interrupt および r_<Config_SCI0>_txif_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void        r_<Config_SCI0>_callback_transmitend ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_callback_receiveend

【 SCI/RSCI 】受信データフル割り込みの発生に伴う処理を行います。

備考 1. 本 API 関数は、r_<Config_SCI0>_receive_interrupt のコールバック・ルーチンとして呼び出されます。

【 SCIF 】受信 FIFO データフル割り込みの発生に伴う処理を行います。

備考 2. 本 API 関数は、r_<Config_SCI0>_rxif_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void        r_<Config_SCI0>_callback_receiveend ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_callback_receiveerror

【 SCI/RSCI 】受信エラー割り込みの発生に伴う処理を行います。

備考　本 API 関数は、r_<Config_SCI0>_receiveerror_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void    r_<Config_SCI0>_callback_receiveerror ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_callback_error

【 SCIF 】受信エラー／ブレーク割り込みの発生に伴う処理を行います。

備考　本 API 関数は、r_<Config_SCI0>_erif_interrupt および r_<Config_SCI0>_brif_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	r_<Config_SCI0>_callback_error (scif_error_type_t *error_type*);

[引数]

I/O	引数	説明
I	scif_error_type_t *error_type*;	割り込みの発生要因 RECEIVE_ERROR : フレーミングエラーまたはパリティエラー OVERRUN_ERROR : オーバラン BREAK_DETECT : ブレーク

[戻り値]

なし

使用例

大文字'A'を送信する

main.c

```
#include "r_smc_entry.h"
extern volatile uint8_t g_sci0_tx_buf;
void main(void)
{
    /* Start the SCI0 channel */
    R_Config_SCI0_Start();

    /* Transmit SCI0 data */
    R_Config_SCI0_Serial_Send((uint8_t *)&g_sci0_tx_buf, 1U);

    while (1U)
    {
        nop();
    }
}
```

Config_SCI0_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_sci0_tx_buf;
/* End user code. Do not edit comment generated here */

void R_Config_SCI0_Create_UserInit(void)
{
    /* Start user code for user init. Do not edit comment generated here */
    g_sci0_tx_buf = 'A';
    /* End user code. Do not edit comment generated here */
}
static void r_Config_SCI0_callback_transmitend(void)
{
    /* Start user code for r_Config_SCI0_callback_transmitend. Do not edit comment generated
here */
    /* Stop the SCI0 channel */
    R_Config_SCI0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

4.2.32 SCI/SCIF クロック同期式モード

以下に、コード生成ツールが SCI/SCIF クロック同期式モード用として出力する API 関数の一覧を示します。

表 4.33 SCI/SCIF/RSCI クロック同期式モード用 API 関数

API 関数名	機能概要
R_<Config_SCI0>_Create	SCI/SCIF クロック同期式モードを制御するうえで必要となる初期化処理を行います。
R_<Config_SCI0>_Start	通信を開始します。
R_<Config_SCI0>_Stop	通信を終了します。
R_<Config_SCI0>_Serial_Send	送信を開始します。（調歩同期式モード）
R_<Config_SCI0>_Serial_Receive	受信を開始します。（調歩同期式モード）
R_<Config_SCI0>_Serial_Send_Receive	送受信を開始します。（クロック同期式モード）
R_<Config_SCI0>_Create_UserInit	SCI/SCIF クロック同期式モードに関するユーザ独自の初期化処理を行います。
r_<Config_SCI0>_transmitend_interrupt	【 SCI/RSCI 】送信終了割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_transmit_interrupt	【 SCI/RSCI 】送信データエンプティ割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_receive_interrupt	【 SCI/RSCI 】受信データフル割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_receiveerror_interrupt	【 SCI/RSCI 】受信エラー割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_teif_interrupt	【 SCIF 】送信完了割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_txif_interrupt	【 SCIF 】送信 FIFO データエンプティ割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_rxif_interrupt	【 SCIF 】受信 FIFO データフル割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_erif_interrupt	【 SCIF 】受信エラー割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_brif_interrupt	【 SCIF 】ブレーク割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_callback_transmitend	【 SCI/RSCI 】送信終了／送信データエンプティ割り込みの発生に伴う処理を行います。 【 SCIF 】送信完了／送信 FIFO データエンプティ割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_callback_receiveend	【 SCI/RSCI 】受信データフル割り込みの発生に伴う処理を行います。 【 SCIF 】受信 FIFO データフル割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_callback_receiveerror	【 SCI/RSCI 】受信エラー割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_callback_error	【 SCIF 】受信エラー／ブレーク割り込みの発生に伴う処理を行います。

R_<Config_SCI0>_Create

SCI/SCIF クロック同期式モードを制御するうえで必要となる初期化処理を行います。

備考 本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void        R_<Config_SCI0>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_SCI0>_Start

通信を開始します。

[指定形式]

```
void        R_<Config_SCI0>_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_SCI0>_Stop

通信を終了します。

[指定形式]

```
void        R_<Config_SCI0>_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_SCI0>_Serial_Send

送信を開始します。(調歩同期式モード)

備考 1. 本 API 関数では、引数 *tx_buf* で指定されたバッファから 1 バイト単位の送信を引数 *tx_num* で指定された回数だけ繰り返し行います。

備考 2. 送信を行う際には、本 API 関数の呼び出し以前に R_<Config_SCI0>_Start を呼び出す必要があります。

[指定形式]

MD_STATUS	R_<Config_SCI0>_Serial_Send (uint8_t * const *tx_buf*, uint16_t *tx_num*);

[引数]

I/O	引数	説明
I	uint8_t * const *tx_buf*;	送信するデータを格納したバッファへのポインタ
I	uint16_t *tx_num*;	送信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数 *tx_num* の指定が不正

R_<Config_SCI0>_Serial_Receive

受信を開始します。(調歩同期式モード)

備考 1. 本 API 関数では、 1 バイト単位の受信を引数 *rx_num* で指定された回数だけ繰り返し行い、引数 *rx_buf* で指定されたバッファに格納します。

備考 2. 受信を行う際には、本 API 関数の呼び出し以前に R_<Config_SCI0>_Start を呼び出す必要があります。

[指定形式]

```
MD_STATUS      R_<Config_SCI0>_Serial_Receive ( uint8_t * const rx_buf, uint16_t rx_num );
```

[引数]

I/O	引数	説明
O	uint8_t * const *rx_buf*;	受信したデータを格納するバッファへのポインタ
I	uint16_t *rx_num*;	受信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数 *rx_num* の指定が不正

R_<Config_SCI0>_Serial_Send_Receive

送受信を開始します。（クロック同期式モード）

備考 1. 本 API 関数では、送信処理として、引数 *tx_buf* で指定されたバッファから 1 バイト単位の送信を引数 *tx_num* で指定された回数だけ繰り返し行います。

備考 2. 本 API 関数では、受信処理として、 1 バイト単位の受信を引数 *rx_num* で指定された回数だけ繰り返し行い、引数 *rx_buf* で指定されたバッファに格納します。

備考 3. 受信を行う際には、本 API 関数の呼び出し以前に R_<Config_SCI0>_Start を呼び出す必要があります。

[指定形式]

MD_STATUS R_<Config_SCI0>_Serial_Send_Receive (uint8_t * const *tx_buf*, uint16_t *tx_num*, uint8_t * const *rx_buf*, uint16_t *rx_num*);

[引数]

I/O	引数	説明
I	uint8_t * const *tx_buf*;	送信するデータを格納したバッファへのポインタ
I	uint16_t *tx_num*;	送信するデータの総数
O	uint8_t * const *rx_buf*;	受信したデータを格納するバッファへのポインタ
I	uint16_t *rx_num*;	受信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数 *tx_num* の指定が不正

R_<Config_SCI0>_Create_UserInit

SCI/SCIF クロック同期式モードに関するユーザ独自の初期化処理を行います。

備考 本 API 関数は、R_<Config_SCI0>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_SCI0>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_transmitend_interrupt

【 SCI/RSCI 】送信終了割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_SCI0>_transmitend_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_transmit_interrupt

【 SCI/RSCI 】送信データエンプティ割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_SCI0>_transmit_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_receive_interrupt

【 SCI/RSCI 】受信データフル割り込みの発生に伴う処理を行います。

このハンドラ内で受信割込み要求禁止(SCI0.SCR.BIT.RIE=0 / RSCI10.SCR0.BIT.RIE=0) を行っています。これが問題となる場合は、お客様の判断でソースコードの編集を行ってください。

[指定形式]

```
void        r_<Config_SCI0>_receive_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_receiveerror_interrupt

【 SCI/RSCI 】受信エラー割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_SCI0>_receiveerror_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_teif_interrupt

【 SCIF 】送信完了割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_SCI0>_teif_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_txif_interrupt

【 SCIF 】送信 FIFO データエンプティ割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_SCI0>_txif_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_rxif_interrupt

【 SCIF 】受信 FIFO データフル割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_SCI0>_rxif_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_erif_interrupt

【 SCIF 】受信エラー割り込みの発生に伴う処理を行います。

備考　本 API 関数は、フレーミングエラーまたはパリティエラーによる割り込みに対応した割り込み処理として呼び出されます。

[指定形式]

```
void    r_<Config_SCI0>_erif_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_brif_interrupt

【 SCIF 】ブレーク割り込みの発生に伴う処理を行います。

備考　本 API 関数は、ブレークまたはオーバランによる割り込みに対応した割り込み処理として呼び出されます。

[指定形式]

```
void    r_<Config_SCI0>_brif_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_callback_transmitend

【 SCI/RSCI 】送信終了／送信データエンプティ割り込みの発生に伴う処理を行います。

備考 1. 本 API 関数は、r_<Config_SCI0>_transmitend_interrupt および r_<Config_SCI0>_transmit_interrupt のコールバック・ルーチンとして呼び出されます。

【 SCIF 】送信完了／送信 FIFO データエンプティ割り込みの発生に伴う処理を行います。

備考 2. 本 API 関数は、r_<Config_SCI0>_teif_interrupt および r_<Config_SCI0>_txif_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void        r_<Config_SCI0>_callback_transmitend ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_callback_receiveend

【 SCI/RSCI 】受信データフル割り込みの発生に伴う処理を行います。

備考 1. 本 API 関数は、r_<Config_SCI0>_receive_interrupt のコールバック・ルーチンとして呼び出されます。

【 SCIF 】受信 FIFO データフル割り込みの発生に伴う処理を行います。

備考 2. 本 API 関数は、r_<Config_SCI0>_rxif_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void        r_<Config_SCI0>_callback_receiveend ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_callback_receiveerror

【 SCI/RSCI 】受信エラー割り込みの発生に伴う処理を行います。

備考　本 API 関数は、r_<Config_SCI0>_receiveerror_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	r_<Config_SCI0>_callback_receiveerror (void);

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_callback_error

【 SCIF 】受信エラー／ブレーク割り込みの発生に伴う処理を行います。

備考　本 API 関数は、r_<Config_SCI0>_erif_interrupt および r_<Config_SCI0>_brif_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	r_<Config_SCI0>_callback_error (scif_error_type_t *error_type*);

[引数]

I/O	引数	説明
I	scif_error_type_t *error_type*;	割り込みの発生要因 RECEIVE_ERROR : フレーミングエラーまたはパリティエラー OVERRUN_ERROR : オーバラン BREAK_DETECT : ブレーク

[戻り値]

なし

使用例

受信したデータを送信する

main.c

```
#include "r_smc_entry.h"
extern volatile uint8_t g_sci0_tx_buf;
extern volatile uint8_t g_sci0_rx_buf;
void main(void)
{
    /* Start the SCI0 channel */
    R_Config_SCI0_Start();

    /* Transmit and receive SCI0 data */
    R_Config_SCI0_Serial_Send_Receive((uint8_t *)&g_sci0_tx_buf, 1U, (uint8_t *)&g_sci0_rx_buf, 1U);

    while (1U)
    {
        nop();
    }
}
```

Config_SCI0_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_sci0_tx_buf;
volatile uint8_t g_sci0_rx_buf;
/* End user code. Do not edit comment generated here */

void R_Config_SCI0_Create_UserInit(void)
{
    /* Start user code for user init. Do not edit comment generated here */
    g_sci0_tx_buf = 0U;
    g_sci0_rx_buf = 0U;
    /* End user code. Do not edit comment generated here */
}

static void r_Config_SCI0_callback_transmitend(void)
{
    /* Start user code for r_Config_SCI0_callback_transmitend. Do not edit comment generated
here */
    /* Transmit and receive SCI0 data */
    R_Config_SCI0_Serial_Send_Receive((uint8_t *)&g_sci0_tx_buf, 1U, (uint8_t *)&g_sci0_rx_buf, 1U);
    /* End user code. Do not edit comment generated here */
}

static void r_Config_SCI0_callback_receiveend(void)
{
    /* Start user code for r_Config_SCI0_callback_receiveend. Do not edit comment generated
here */
    g_sci0_tx_buf = g_sci0_rx_buf;
    g_sci0_rx_buf = 0U;
    /* End user code. Do not edit comment generated here */
}
```

4.2.33 シングルスキャンモード S12AD

以下に、コード生成ツールがシングルスキャンモード S12AD 用として出力する API 関数の一覧を示します。

表 4.34 シングルスキャンモード S12AD 用 API 関数

API 関数名	機能概要
R_<Config_S12AD0>_Create	シングルスキャンモード S12AD を制御するうえで必要となる初期化処理を行います。
R_<Config_S12AD0>_Start	A/D 変換を開始します。
R_<Config_S12AD0>_Stop	A/D 変換を終了します。
R_<Config_S12AD0>_Get_ValueResult	変換結果を獲得します。
R_<Config_S12AD0>_Set_CompareValue	コンペアレベルの設定を行います。
R_<Config_S12AD0>_Set_CompareAValue	ウィンドウ A コンペアレベルの設定を行います。
R_<Config_S12AD0>_Set_CompareBValue	ウィンドウ B コンペアレベルの設定を行います。
R_<Config_S12AD0>_Create_UserInit	シングルスキャンモード S12AD に関するユーザ独自の初期化処理を行います。
r_<Config_S12AD0>_interrupt	スキャン終了割り込みの発生に伴う処理を行います。
r_<Config_S12AD0>_compare_interrupt	コンペア割り込みの発生に伴う処理を行います。
r_<Config_S12AD0>_compare_interruptA	ウィンドウ A コンペア割り込みの発生に伴う処理を行います。
r_<Config_S12AD0>_compare_interruptB	ウィンドウ B コンペア割り込みの発生に伴う処理を行います。

R_<Config_S12AD0>_Create

シングルスキャンモード S12AD を制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void    R_<Config_S12AD0>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_S12AD0>_Start

A/D 変換を開始します。

[指定形式]

```
void        R_<Config_S12AD0>_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_S12AD0>_Stop

A/D 変換を終了します。

[指定形式]

void	R_<Config_S12AD0>_Stop (void);

[引数]

なし

[戻り値]

なし

R_<Config_S12AD0>_Get_ValueResult

変換結果を獲得します。

[指定形式]

```
void    R_<Config_S12AD0>_Get_ValueResult ( ad_channel_t channel, uint16_t * const buffer );
```

[引数]

RX130，RX230/RX231 の場合

I/O	引数	説明
I	ad_channel_t *channel*;	チャネル番号 ADCHANNEL0 ：入力チャネル AN000 ADCHANNEL1 ：入力チャネル AN001 ADCHANNEL2 ：入力チャネル AN002 ADCHANNEL3 ：入力チャネル AN003 ADCHANNEL4 ：入力チャネル AN004 ADCHANNEL5 ：入力チャネル AN005 ADCHANNEL6 ：入力チャネル AN006 ADCHANNEL7 ：入力チャネル AN007 ADCHANNEL16 ：入力チャネル AN016 ADCHANNEL17 ：入力チャネル AN017 ADCHANNEL18 ：入力チャネル AN018 ADCHANNEL19 ：入力チャネル AN019 ADCHANNEL20 ：入力チャネル AN020 ADCHANNEL21 ：入力チャネル AN021 ADCHANNEL22 ：入力チャネル AN022 ADCHANNEL23 ：入力チャネル AN023 ADCHANNEL24 ：入力チャネル AN024 ADCHANNEL25 ：入力チャネル AN025 ADCHANNEL26 ：入力チャネル AN026 ADCHANNEL27 ：入力チャネル AN027 ADCHANNEL28 ：入力チャネル AN028 ADCHANNEL29 ：入力チャネル AN029 ADCHANNEL30 ：入力チャネル AN030 ADCHANNEL31 ：入力チャネル AN031 ADTEMPSENSOR ：拡張アナログ入力（温度センサ出力） ADINTERREFVOLT ：拡張アナログ入力（内部基準電圧） ADSELFDIAGNOSIS ：自己診断結果 ADDATADUPLICATION ：ダブルトリガモード結果
O	uint16_t * const *buffer*;	獲得した変換結果を格納する領域へのポインタ

そのほかのデバイスの場合

I/O	引数	説明
I	ad_channel_t *channel*;	チャネル番号 ADCHANNEL0 : 入力チャネル AN000 ADCHANNEL1 : 入力チャネル AN001 ADCHANNEL2 : 入力チャネル AN002 ADCHANNEL3 : 入力チャネル AN003 ADCHANNEL4 : 入力チャネル AN004 ADCHANNEL5 : 入力チャネル AN005 ADCHANNEL6 : 入力チャネル AN006 ADCHANNEL7 : 入力チャネル AN007 ADCHANNEL8 : 入力チャネル AN008 ADCHANNEL9 : 入力チャネル AN009 ADCHANNEL10 : 入力チャネル AN010 ADCHANNEL11 : 入力チャネル AN011 ADCHANNEL12 : 入力チャネル AN012 ADCHANNEL13 : 入力チャネル AN013 ADCHANNEL14 : 入力チャネル AN014 ADCHANNEL15 : 入力チャネル AN015 ADCHANNEL16 : 入力チャネル AN016 ADCHANNEL17 : 入力チャネル AN017 ADCHANNEL18 : 入力チャネル AN018 ADCHANNEL19 : 入力チャネル AN019 ADCHANNEL20 : 入力チャネル AN020 ADTEMPSENSOR : 拡張アナログ入力（温度センサ出力） ADINTERREFVOLT : 拡張アナログ入力（内部基準電圧） ADSELFDIAGNOSIS : 自己診断結果 ADDATADUPLICATION : ダブルトリガモード結果 ADDATADUPLICATIONA : ダブルトリガモード A 結果 ADDATADUPLICATIONB : ダブルトリガモード B 結果
O	uint16_t * const *buffer*;	獲得した変換結果を格納する領域へのポインタ

[戻り値]

なし

R_<Config_S12AD0>_Set_CompareValue

コンペアレベルの設定を行います。

[指定形式]

```
void      R_<Config_S12AD0>_Set_CompareValue ( uint16_t reg_value0, uint16_t reg_value1);
```

[引数]

I/O	引数	説明
I	uint16_t *reg_value0*;	コンペアレジスタ 0 に設定する値
I	uint16_t *reg_value1*;	コンペアレジスタ 1 に設定する値

[戻り値]

なし

R_<Config_S12AD0>_Set_CompareAValue

ウィンドウ A コンペアレベルの設定を行います。

[指定形式]

void R_<Config_S12AD0>_Set_CompareAValue (uint16_t *reg_value0*, uint16_t *reg_value1*);

[引数]

I/O	引数	説明
I	uint16_t *reg_value0*;	コンペアレジスタ 0 に設定する値
I	uint16_t *reg_value1*;	コンペアレジスタ 1 に設定する値

[戻り値]

なし

R_<Config_S12AD0>_Set_CompareBValue

ウィンドウ B コンペアレベルの設定を行います。

[指定形式]

void R_<Config_S12AD0>_Set_CompareBValue (uint16_t *reg_value0*, uint16_t *reg_value1*);

[引数]

I/O	引数	説明
I	uint16_t *reg_value0*;	コンペアレジスタ 0 に設定する値
I	uint16_t *reg_value1*;	コンペアレジスタ 1 に設定する値

[戻り値]

なし

R_<Config_S12AD0>_Create_UserInit

シングルスキャンモード S12AD に関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_S12AD0>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void    R_<Config_S12AD0>_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_S12AD0>_interrupt

スキャン終了割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_S12AD0>_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_S12AD0>_compare_interrupt

コンペア割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_S12AD0>_compare_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_S12AD0>_compare_interruptA

ウィンドウ A コンペア割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_S12AD0>_compare_interruptA ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_S12AD0>_compare_interruptB

ウィンドウ B コンペア割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_S12AD0>_compare_interruptB ( void );
```

[引数]

なし

[戻り値]

なし

使用例

AD 変換結果を取得する

main.c

```
#include "r_smc_entry.h"
void main(void)
{
    /* Start the AD0 converter */
    R_Config_S12AD0_Start();

    while (1U)
    {
        nop();
    }
}
```

Config_S12AD0_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint16_t g_s12ad0_ch000_value;
/* End user code. Do not edit comment generated here */

static void r_Config_S12AD0_interrupt(void)
{
    /* Start user code for r_Config_S12AD0_interrupt. Do not edit comment generated here */
    R_Config_S12AD0_Get_ValueResult(ADCHANNEL0, (uint16_t *)&g_s12ad0_ch000_value);
    /* End user code. Do not edit comment generated here */
}
```

4.2.34 スマートカードインタフェース

以下に、コード生成ツールがスマートカードインタフェース用として出力する API 関数の一覧を示します。

表 4.35 SCI/RSCI スマートカードインタフェース用 API 関数

API 関数名	機能概要
R_<Config_SCI0>_Create	スマートカードインタフェースを制御するうえで必要となる初期化処理を行います。
R_<Config_SCI0>_Start	通信を開始します。
R_<Config_SCI0>_Stop	通信を終了します。
R_<Config_SCI0>_SmartCard_Send	送信を開始します。（スマートカードインタフェースモード）
R_<Config_SCI0>_SmartCard_Receive	受信を開始します。（スマートカードインタフェースモード）
R_<Config_SCI0>_Create_UserInit	スマートカードインタフェースに関するユーザ独自の初期化処理を行います。
r_<Config_SCI0>_transmit_interrupt	送信データエンプティ割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_receive_interrupt	受信データフル割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_receiveerror_interrupt	受信エラー割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_callback_transmitend	送信データエンプティ割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_callback_receiveend	受信データフル割り込みの発生に伴う処理を行います。
r_<Config_SCI0>_callback_receiveerror	受信エラー割り込みの発生に伴う処理を行います。

R_<Config_SCI0>_Create

スマートカードインタフェースを制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void    R_<Config_SCI0>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_SCI0>_Start

通信を開始します。

[指定形式]

```
void        R_<Config_SCI0>_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_SCI0>_Stop

通信を終了します。

[指定形式]

```
void        R_<Config_SCI0>_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_SCI0>_SmartCard_Send

送信を開始します。（スマートカードインタフェースモード）

備考 1. 本 API 関数では、引数 *tx_buf* で指定されたバッファから 1 バイト単位の送信を引数 *tx_num* で指定された回数だけ繰り返し行います。

備考 2. 送信を行う際には、本 API 関数の呼び出し以前に R_<Config_SCI0>_Start を呼び出す必要があります。

[指定形式]

```
MD_STATUS      R_<Config_SCI0>_SmartCard_Send ( uint8_t * const tx_buf, uint16_t tx_num );
```

[引数]

I/O	引数	説明
I	uint8_t * const *tx_buf*;	送信するデータを格納したバッファへのポインタ
I	uint16_t *tx_num*;	送信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数 *tx_num* の指定が不正

R_<Config_SCI0>_SmartCard_Receive

受信を開始します。（スマートカードインタフェースモード）

備考 1. 本 API 関数では、 1 バイト単位の受信を引数 *rx_num* で指定された回数だけ繰り返し行い、引数 *rx_buf* で指定されたバッファに格納します。

備考 2. 受信を行う際には、本 API 関数の呼び出し以前に R_<Config_SCI0>_Start を呼び出す必要があります。

[指定形式]

```
MD_STATUS    R_<Config_SCI0>_SmartCard_Receive ( uint8_t * const rx_buf, uint16_t rx_num );
```

[引数]

I/O	引数	説明
O	uint8_t * const *rx_buf*;	受信したデータを格納するバッファへのポインタ
I	uint16_t *rx_num*;	受信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数 *rx_num* の指定が不正

R_<Config_SCI0>_Create_UserInit

スマートカードインタフェースに関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_SCI0>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_SCI0>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_transmit_interrupt

送信バッファエンプティ割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_SCI0>_transmit_interrupt (void);

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_receive_interrupt

受信バッファフル割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_SCI0>_receive_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_receiveerror_interrupt

受信エラー割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_SCI0>_receiveerror_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_callback_transmitend

送信バッファエンプティ割り込みの発生に伴う処理を行います。

備考　本 API 関数は、r_<Config_SCI0>_transmit_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void    r_<Config_SCI0>_callback_transmitend ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_callback_receiveend

受信バッファフル割り込みの発生に伴う処理を行います。

備考　本 API 関数は、r_<Config_SCI0>_receive_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void    r_<Config_SCI0>_callback_receiveend ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_SCI0>_callback_receiveerror

受信エラー割り込みの発生に伴う処理を行います。

備考 本 API 関数は、r_<Config_SCI0>_receiveerror_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void r_<Config_SCI0>_callback_receiveerror ( void );
```

[引数]

なし

[戻り値]

なし

使用例

1Byte 送信後、1Byte 受信に切り替える

main.c

```
#include "r_smc_entry.h"
extern volatile uint8_t g_sci0_tx_buf;
void main(void)
{
    /* Start the SCI0 channel */
    R_Config_SCI0_Start();

    /* Transmit SCI0 data */
    R_Config_SCI0_SmartCard_Send((uint8_t *)&g_sci0_tx_buf, 1U);

    while (1U)
    {
        nop();
    }
}
```

Config_SCI0_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_sci0_tx_buf;
volatile uint8_t g_sci0_rx_buf;
/* End user code. Do not edit comment generated here */

void R_Config_SCI0_Create_UserInit(void)
{
    /* Start user code for user init. Do not edit comment generated here */
    g_sci0_tx_buf = 'A';
    /* End user code. Do not edit comment generated here */
}

static void r_Config_SCI0_callback_transmitend(void)
{
    /* Start user code for r_<onfig_SCI0_callback_transmitend. Do not edit comment generated here */
    /* Stop the SCI0 channel */
    R_Config_SCI0_Stop();

    /* Initializes SCI0 */
    R_Config_SCI0_Create();

    /* Start the SCI0 channel */
    R_Config_SCI0_Start();

    /* Receive SCI0 data */
    R_Config_SCI0_SmartCard_Receive((uint8_t *)&g_sci0_rx_buf, 1U);
    /* End user code. Do not edit comment generated here */
}
static void r_Config_SCI0_callback_receiveend(void)
{
    /* Start user code for r_Config_SCI0_callback_receiveend. Do not edit comment generated here */
    /* Stop the SCI0 channel */
    R_Config_SCI0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

4.2.35 SPI クロック同期式モード

以下に、コード生成ツールが SPI クロック同期式モード（RSPI/SCI/RSCI）用として出力する API 関数の一覧を示します。

表 4.36 SPI クロック同期式モード（RSPI/SCI/RSCI）用 API 関数

API 関数名	機能概要
R_<Config_RSPI0>_Create	SPI クロック同期式モード（RSPI/SCI/RSCI）を制御するうえで必要となる初期化処理を行います。
R_<Config_RSPI0>_Start	通信を開始します。
R_<Config_RSPI0>_Stop	通信を終了します。
R_<Config_RSPI0>_Send	【 RSPI 】送信を開始します。
R_<Config_RSPI0>_Send_Receive	【 RSPI 】送受信を開始します。
R_<Config_RSPI0>_SPI_Master_Send	【 SCI/RSCI 】マスタ送信を開始します。（簡易 SPI モード）
R_<Config_RSPI0>_SPI_Master_Send_Receive	【 SCI/RSCI 】マスタ送受信を開始します。（簡易 SPI モード）
R_<Config_RSPI0>_SPI_Slave_Send	【 SCI/RSCI 】スレーブ送信を開始します。（簡易 SPI モード）
R_<Config_RSPI0>_SPI_Slave_Send_Receive	【 SCI/RSCI 】スレーブ送受信を開始します。（簡易 SPI モード）
R_<Config_RSPI0>_Create_UserInit	SPI クロック同期式モード（RSPI/SCI/RSCI）に関するユーザ独自の初期化処理を行います。
r_<Config_RSPI0>_receive_interrupt	受信バッファフル割り込みの発生に伴う処理を行います。
r_<Config_RSPI0>_transmit_interrupt	送信バッファエンプティ割り込みの発生に伴う処理を行います。
r_<Config_RSPI0>_error_interrupt	【 RSPI 】エラー割り込みの発生に伴う処理を行います。
r_<Config_RSPI0>_idle_interrupt	【 RSPI 】アイドル割り込みの発生に伴う処理を行います。
r_<Config_RSPI0>_transmitend_interrupt	【 SCI/RSCI 】送信終了割り込みの発生に伴う処理を行います。
r_<Config_RSPI0>_receiveerror_interrupt	【 SCI/RSCI 】受信エラー割り込みの発生に伴う処理を行います。
r_<Config_RSPI0>_callback_receiveend	受信バッファフル割り込みの発生に伴う処理を行います。
r_<Config_RSPI0>_callback_transmitend	送信バッファエンプティ割り込みの発生に伴う処理を行います。 【 RSPI 】アイドル割り込みの発生に伴う処理を行います。 【 SCI/RSCI 】送信終了割り込みの発生に伴う処理を行います。
r_<Config_RSPI0>_callback_error	【 RSPI 】エラー割り込みの発生に伴う処理を行います。
r_<Config_RSPI0>_callback_receiveerror	【 SCI/RSCI 】受信エラー割り込みの発生に伴う処理を行います。

R_<Config_RSPI0>_Create

SPI クロック同期式モード（RSPI/SCI/RSCI）を制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void    R_<Config_RSPI0>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_RSPI0>_Start

通信を開始します。

[指定形式]

```
void        R_<Config_RSPI0>_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_RSPI0>_Stop

通信を終了します。

[指定形式]

```
void        R_<Config_RSPI0>_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_RSPI0>_Send

【 RSPI 】送信を開始します。

備考 1. 本 API 関数では、引数 *tx_buf* で指定されたバッファから 1 バイト単位の送信を引数 *tx_num* で指定された回数だけ繰り返し行います。

備考 2. 送信を行う際には、本 API 関数の呼び出し以前に R_<Config_RSPI0>_Start を呼び出す必要があります。

[指定形式]

MD_STATUS	R_<Config_RSPI0>_Send (type * const *tx_buf*, uint16_t *tx_num*);

[引数]

I/O	引数	説明
I	type* const *tx_buf*;	送信するデータを格納したバッファへのポインタ。「Type」は GUI で設定したバッファアクセス幅に合わせて、コード生成時に「uint32_t」、「uint16_t」、もしくは「uint8_t」に変更されます。
I	uint16_t *tx_num*;	送信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数 *tx_num* の指定が不正

R_<Config_RSPI0>_Send_Receive

【 RSPI 】送受信を開始します。

備考 1. 本 API 関数では、送信処理として、引数 *tx_buf* で指定されたバッファから 1 バイト単位の送信を引数 *tx_num* で指定された回数だけ繰り返し行います。

備考 2. 本 API 関数では、受信処理として、 1 バイト単位の受信を引数 *tx_num* で指定された回数だけ繰り返し行い、引数 *rx_buf* で指定されたバッファに格納します。

備考 3. 送受信を行う際には、本 API 関数の呼び出し以前に R_<Config_RSPI0>_Start を呼び出す必要があります。

[指定形式]

```
MD_STATUS      R_<Config_RSPI0>_Send_Receive ( type * const tx_buf, uint16_t tx_num,
 type * const rx_buf );
```

[引数]

I/O	引数	説明
I	type * const *tx_buf*;	送信するデータを格納したバッファへのポインタ。「Type」は GUI で設定したバッファアクセス幅に合わせて、コード生成時に「uint32_t」、「uint16_t」、もしくは「uint8_t」に変更されます。
I	uint16_t *tx_num*;	送受信するデータの総数
O	type * const *rx_buf*;	受信したデータを格納するバッファへのポインタ。「Type」は GUI で設定したバッファアクセス幅に合わせて、コード生成時に「uint32_t」、「uint16_t」、もしくは「uint8_t」に変更されます。

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数 *tx_num* の指定が不正

R_<Config_RSPI0>_SPI_Master_Send

【 SCI/RSCI 】マスタ送信を開始します。（簡易 SPI モード）

備考 1. 本 API 関数では、引数 *tx_buf* で指定されたバッファから 1 バイト単位のマスタ送信を引数 *tx_num* で指定された回数だけ繰り返し行います。

備考 2. マスタ送信を行う際には、本 API 関数の呼び出し以前に R_<Config_RSPI0>_Start を呼び出す必要があります。

[指定形式]

```
MD_STATUS      R_<Config_RSPI0>_SPI_Master_Send (type * const tx_buf, uint16_t tx_num );
```

[引数]

I/O	引数	説明
I	type * const *tx_buf*;	送信するデータを格納したバッファへのポインタ。「Type」は GUI で設定したバッファアクセス幅に合わせて、コード生成時に「uint32_t」、「uint16_t」、もしくは「uint8_t」に変更されます。
I	uint16_t *tx_num*;	送信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数 *tx_num* の指定が不正

R_<Config_RSPI0>_SPI_Master_Send_Receive

【 SCI/RSCI 】マスタ送受信を開始します。（簡易 SPI モード）

備考 1. 本 API 関数では、マスタ送信処理として、引数 *tx_buf* で指定されたバッファから 1 バイト単位のマスタ送信を引数 *tx_num* で指定された回数だけ繰り返し行います。

備考 2. 本 API 関数では、マスタ受信処理として、1 バイト単位のマスタ受信を引数 *rx_num* で指定された回数だけ繰り返し行い、引数 *rx_buf* で指定されたバッファに格納します。

備考 3. 送受信を行う際には、本 API 関数の呼び出し以前に R_<Config_RSPI0>_Start を呼び出す必要があります。

[指定形式]

```
MD_STATUS      R_<Config_RSPI0>_SPI_Master_Send_Receive ( type * const tx_buf, uint16_t tx_num, type_t * const rx_buf, uint16_t rx_num );
```

[引数]

I/O	引数	説明
I	type * const *tx_buf*;	送信するデータを格納したバッファへのポインタ。「Type」は GUI で設定したバッファアクセス幅に合わせて、コード生成時に「uint32_t」、「uint16_t」、もしくは「uint8_t」に変更されます。
I	uint16_t *tx_num*;	送信するデータの総数
O	type * const *rx_buf*;	受信したデータを格納するバッファへのポインタ。「Type」は GUI で設定したバッファアクセス幅に合わせて、コード生成時に「uint32_t」、「uint16_t」、もしくは「uint8_t」に変更されます。
I	uint16_t *rx_num*;	受信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数 *tx_num* の指定が不正

R_<Config_RSPI0>_SPI_Slave_Send

【 SCI/RSCI 】スレーブ送信を開始します。（簡易 SPI モード）

備考 1. 本 API 関数では、引数 *tx_buf* で指定されたバッファから 1 バイト単位のスレーブ送信を引数 *tx_num* で指定された回数だけ繰り返し行います。

備考 2. スレーブ送信を行う際には、本 API 関数の呼び出し以前に R_<Config_RSPI0>_Start を呼び出す必要があります。

[指定形式]

```
MD_STATUS      R_<Config_RSPI0>_SPI_Slave_Send (type * const tx_buf, uint16_t tx_num );
```

[引数]

I/O	引数	説明
I	type * const *tx_buf*;	送信するデータを格納したバッファへのポインタ。「Type」は GUI で設定したバッファアクセス幅に合わせて、コード生成時に「uint32_t」、「uint16_t」、もしくは「uint8_t」に変更されます。
I	uint16_t *tx_num*;	送信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数 *tx_num* の指定が不正

R_<Config_RSPI0>_SPI_Slave_Send_Receive

【 SCI/RSCI 】スレーブ送受信を開始します。（簡易 SPI モード）

備考 1. 本 API 関数では、スレーブ送信処理として、引数 *tx_buf* で指定されたバッファから 1 バイト単位のスレーブ送信を引数 *tx_num* で指定された回数だけ繰り返し行います。

備考 2. 本 API 関数では、スレーブ受信処理として、 1 バイト単位のスレーブ受信を引数 *rx_num* で指定された回数だけ繰り返し行い、引数 *rx_buf* で指定されたバッファに格納します。

備考 3. 送受信を行う際には、本 API 関数の呼び出し以前に R_<Config_RSPI0>_Start を呼び出す必要があります。

[指定形式]

```
MD_STATUS      R_<Config_RSPI0>_SPI_Slave_Send_Receive ( type * const tx_buf,
 uint16_t tx_num, type * const rx_buf, uint16_t rx_num );
```

[引数]

I/O	引数	説明
I	type * const *tx_buf*;	送信するデータを格納したバッファへのポインタ。「Type」は GUI で設定したバッファアクセス幅に合わせて、コード生成時に「uint32_t」、「uint16_t」、もしくは「uint8_t」に変更されます。
I	uint16_t *tx_num*;	送信するデータの総数
O	type * const *rx_buf*;	受信したデータを格納するバッファへのポインタ。「Type」は GUI で設定したバッファアクセス幅に合わせて、コード生成時に「uint32_t」、「uint16_t」、もしくは「uint8_t」に変更されます。
I	uint16_t *rx_num*;	受信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数 *tx_num* の指定が不正

R_<Config_RSPI0>_Create_UserInit

SPI クロック同期式モード（RSPI/SCI/RSCI）に関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_RSPI0>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void    R_<Config_RSPI0>_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_RSPI0>_receive_interrupt

受信バッファフル割り込みの発生に伴う処理を行います。

このハンドラ内で受信割込み要求禁止(SCI0.SCR.BIT.RIE=0 / RSCI10.SCR0.BIT.RIE=0) または受信バッファフル割込み要求禁止(RSPI0.SPCR.BIT.SPRIE = 0)を行っています。これが問題となる場合は、お客様の判断でソースコードの編集を行ってください。

[指定形式]

```
void        r_<Config_RSPI0>_receive_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_RSPI0>_transmit_interrupt

送信バッファエンプティ割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_RSPI0>_transmit_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_RSPI0>_error_interrupt

【 RSPI 】エラー割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_RSPI0>_error_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_RSPI0>_idle_interrupt

【 RSPI 】アイドル割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_RSPI0>_idle_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_RSPI0>_transmitend_interrupt

【 SCI/RSCI 】送信終了割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_RSPI0>_transmitend_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_RSPI0>_receiveerror_interrupt

【 SCI/RSCI 】受信エラー割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_RSPI0>_receiveerror_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_RSPI0>_callback_receiveend

受信バッファフル割り込みの発生に伴う処理を行います。

備考　本 API 関数は、r_<Config_RSPI0>_receive_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void    r_<Config_RSPI0>_callback_receiveend ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_RSPI0>_callback_transmitend

送信バッファエンプティ割り込みの発生に伴う処理を行います。

備考 1. 本 API 関数は、r_<Config_RSPI0>_transmit_interrupt のコールバック・ルーチンとして呼び出されます。

【 RSPI 】アイドル割り込みの発生に伴う処理を行います。

備考 2. 本 API 関数は、r_<Config_RSPI0>_idle_interrupt のコールバック・ルーチンとして呼び出されます。

【 SCI/RSCI 】送信終了割り込みの発生に伴う処理を行います。

備考 3. 本 API 関数は、r_<Config_RSPI0>_transmitend_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void        r_<Config_RSPI0>_callback_transmitend ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_RSPI0>_callback_error

【 RSPI 】エラー割り込みの発生に伴う処理を行います。

備考 1. 本 API 関数は、r_<Config_RSPI0>_error_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void        r_<Config_RSPI0>_callback_error ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_RSPI0>_callback_receiveerror

【 SCI/RSCI 】受信エラー割り込みの発生に伴う処理を行います。

備考 1. 本 API 関数は、r_<Config_RSPI0>_receiveerror_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void        r_<Config_RSPI0>_callback_receiveerror ( void );
```

[引数]

なし

[戻り値]

なし

使用例

送信と同じデータ数の受信が完了すると通信を停止する

main.c

```
#include "r_smc_entry.h"
extern volatile uint32_t g_rspi0_tx_buf[4];
extern volatile uint32_t g_rspi0_rx_buf[4];
void main(void)
{
    /* Start the RSPI0 module operation */
    R_Config_RSPI0_Start();

    /* Send and receive RSPI0 data */
    R_Config_RSPI0_Send_Receive((uint32_t *)g_rspi0_tx_buf, 4U, (uint32_t *)g_rspi0_rx_buf);

    while (1U)
    {
        nop();
    }
}
```

Config_RSPI0_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint32_t g_rspi0_tx_buf[4];
volatile uint32_t g_rspi0_rx_buf[4];
/* End user code. Do not edit comment generated here */

void R_Config_RSPI0_Create_UserInit(void)
{
    /* Start user code for user init. Do not edit comment generated here */
    g_rspi0_tx_buf[0] = 0x000000FF;
    g_rspi0_tx_buf[1] = 0x0000FF00;
    g_rspi0_tx_buf[2] = 0x00FF0000;
    g_rspi0_tx_buf[3] = 0xFF000000;
    /* End user code. Do not edit comment generated here */
}

static void r_Config_RSPI0_callback_receiveend(void)
{
    /* Start user code for r_Config_RSPI0_callback_receiveend. Do not edit comment generated
here */
    /* Stop the RSPI0 module operation */
    R_Config_RSPI0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

4.2.36 SPI 動作モード

以下に、コード生成ツールが SPI 動作モード用として出力する API 関数の一覧を示します。

表 4.37 SPI 動作モード用 API 関数

API 関数名	機能概要
R_<Config_RSPI0>_Create	SPI 動作モードを制御するうえで必要となる初期化処理を行います。
R_<Config_RSPI0>_Start	通信を開始します。
R_<Config_RSPI0>_Stop	通信を終了します。
R_<Config_RSPI0>_Send	送信を開始します。
R_<Config_RSPI0>_Send_Receive	送受信を開始します。
R_<Config_RSPI0>_Create_UserInit	SPI 動作モードに関するユーザ独自の初期化処理を行います。
r_<Config_RSPI0>_receive_interrupt	受信バッファフル割り込みの発生に伴う処理を行います。
r_<Config_RSPI0>_transmit_interrupt	送信バッファエンプティ割り込みの発生に伴う処理を行います。
r_<Config_RSPI0>_error_interrupt	エラー割り込みの発生に伴う処理を行います。
r_<Config_RSPI0>_idle_interrupt	アイドル割り込みの発生に伴う処理を行います。
r_<Config_RSPI0>_callback_receiveend	受信バッファフル割り込みの発生に伴う処理を行います。
r_<Config_RSPI0>_callback_transmitend	送信バッファエンプティ／アイドル割り込みの発生に伴う処理を行います。
r_<Config_RSPI0>_callback_error	エラー割り込みの発生に伴う処理を行います。

R_<Config_RSPI0>_Create

SPI 動作モードを制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void    R_<Config_RSPI0>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_RSPI0>_Start

通信を開始します。

[指定形式]

```
void        R_<Config_RSPI0>_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_RSPI0>_Stop

通信を終了します。

[指定形式]

```
void        R_<Config_RSPI0>_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_RSPI0>_Send

送信を開始します。

備考 1. 本 API 関数では、引数 *tx_buf* で指定されたバッファから 1 バイト単位の送信を引数 *tx_num* で指定された回数だけ繰り返し行います。

備考 2. 送信を行う際には、本 API 関数の呼び出し以前に R_<Config_RSPI0>_Start を呼び出す必要があります。

[指定形式]

MD_STATUS R_<Config_RSPI0>_Send (type * const *tx_buf*, uint16_t *tx_num*);

[引数]

I/O	引数	説明
I	type * const *tx_buf*;	送信するデータを格納したバッファへのポインタ。「Type」は GUI で設定したバッファアクセス幅に合わせて、コード生成時に「uint32_t」、「uint16_t」、もしくは「uint8_t」に変更されます。
I	uint16_t *tx_num*;	送信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数 *tx_num* の指定が不正

R_<Config_RSPI0>_Send_Receive

送受信を開始します。

備考 1. 本 API 関数では、送信処理として、引数 *tx_buf* で指定されたバッファから 1 バイト単位の送信を引数 *tx_num* で指定された回数だけ繰り返し行います。

備考 2. 本 API 関数では、受信処理として、 1 バイト単位の受信を引数 *tx_num* で指定された回数だけ繰り返し行い、引数 *rx_buf* で指定されたバッファに格納します。

備考 3. 送受信を行う際には、本 API 関数の呼び出し以前に R_<Config_RSPI0>_Start を呼び出す必要があります。

[指定形式]

```
MD_STATUS      R_<Config_RSPI0>_Send_Receive ( type * const tx_buf, uint16_t tx_num,
 type * const rx_buf );
```

[引数]

I/O	引数	説明
I	type * const *tx_buf*;	送信するデータを格納したバッファへのポインタ。「Type」は GUI で設定したバッファアクセス幅に合わせて、コード生成時に「uint32_t」、「uint16_t」、もしくは「uint8_t」に変更されます。
I	uint16_t *tx_num*;	送受信するデータの総数
O	type * const *rx_buf*;	受信したデータを格納するバッファへのポインタ。「Type」は GUI で設定したバッファアクセス幅に合わせて、コード生成時に「uint32_t」、「uint16_t」、もしくは「uint8_t」に変更されます。

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数 *tx_num* の指定が不正

R_<Config_RSPI0>_Create_UserInit

SPI 動作モードに関するユーザ独自の初期化処理を行います。

備考 本 API 関数は、R_<Config_RSPI0>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_RSPI0>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

r_<Config_RSPI0>_receive_interrupt

受信バッファフル割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_RSPI0>_receive_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_RSPI0>_transmit_interrupt

送信バッファエンプティ割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_RSPI0>_transmit_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_RSPI0>_error_interrupt

エラー割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_RSPI0>_error_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_RSPI0>_idle_interrupt

アイドル割り込みの発生に伴う処理を行います。

[指定形式]

```
void        r_<Config_RSPI0>_idle_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_RSPI0>_callback_receiveend

受信バッファフル割り込みの発生に伴う処理を行います。

備考　本 API 関数は、r_<Config_RSPI0>_receive_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void    r_<Config_RSPI0>_callback_receiveend ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_RSPI0>_callback_transmitend

送信バッファエンプティ／アイドル割り込みの発生に伴う処理を行います。

備考　本 API 関数は、r_<Config_RSPI0>_transmit_interrupt および r_<Config_RSPI0>_idle_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void    r_<Config_RSPI0>_callback_transmitend ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_RSPI0>_callback_error

エラー割り込みの発生に伴う処理を行います。

備考　本 API 関数は、r_<Config_RSPI0>_error_interrupt のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void    r_<Config_RSPI0>_callback_error ( void );
```

[引数]

なし

[戻り値]

なし

使用例

送信と同じデータ数の受信が完了すると通信を停止する

main.c

```
#include "r_smc_entry.h"
extern volatile uint32_t g_rspi0_tx_buf[4];
extern volatile uint32_t g_rspi0_rx_buf[4];
void main(void)
{
    /* Start the RSPI0 module operation */
    R_Config_RSPI0_Start();

    /* Send and receive RSPI0 data */
    R_Config_RSPI0_Send_Receive((uint32_t *)g_rspi0_tx_buf, 4U, (uint32_t *)g_rspi0_rx_buf);

    while (1U)
    {
        nop();
    }
}
```

Config_RSPI0_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint32_t g_rspi0_tx_buf[4];
volatile uint32_t g_rspi0_rx_buf[4];
/* End user code. Do not edit comment generated here */

void R_Config_RSPI0_Create_UserInit(void)
{
    /* Start user code for user init. Do not edit comment generated here */
    g_rspi0_tx_buf[0] = 0x000000FF;
    g_rspi0_tx_buf[1] = 0x0000FF00;
    g_rspi0_tx_buf[2] = 0x00FF0000;
    g_rspi0_tx_buf[3] = 0xFF000000;
    /* End user code. Do not edit comment generated here */
}

static void r_Config_RSPI0_callback_receiveend(void)
{
    /* Start user code for r_Config_RSPI0_callback_receiveend. Do not edit comment generated
here */
    /* Stop the RSPI0 module operation */
    R_Config_RSPI0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

4.2.37 電圧検出回路

以下に、コード生成ツールが電圧検出回路用として出力する API 関数の一覧を示します。

表 4.38 電圧検出回路用 API 関数

API 関数名	機能概要
R_<Config_LVD1>_Create	電圧検出回路を制御するうえで必要となる初期化処理を行います。
R_<Config_LVD1>_Start	電圧の監視を開始します。
R_<Config_LVD1>_Stop	電圧の監視を終了します。
R_<Config_LVD1>_Create_UserInit	電圧検出回路に関するユーザ独自の初期化処理を行います。
r_<Config_LVD1>_lvdn_interrupt	電圧監視割り込みの発生に伴う処理を行います。

R_<Config_LVD1>_Create

電圧検出回路を制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void    R_<Config_LVD1>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_LVD1>_Start

電圧の監視を開始します。

[指定形式]

void	R_<Config_LVD1>_Start (void);

[引数]

なし

[戻り値]

なし

R_<Config_LVD1>_Stop

電圧の監視を終了します。

[指定形式]

```
void        R_<Config_LVD1>_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_LVD1>_Create_UserInit

電圧検出回路に関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_LVD1>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_LVD1>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

r_<Config_LVD1>_lvd*n*_interrupt

電圧監視割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_LVD1>_lvd*n*_interrupt (void);

備考 *n* は回路番号を意味します。

[引数]

なし

[戻り値]

なし

使用例

電圧監視割り込み発生時にソフトウェアリセット

main.c

```
#include "r_smc_entry.h"
void main(void)
{
    /* Start the LVD1 operation */
    R_Config_LVD1_Start();

    while (1U)
    {
        nop();
    }
}
```

Config_LVD1_user.c

```
void r_Config_LVD1_lvd1_interrupt(void)
{
    /* Start user code for r_Config_LVD1_lvd1_interrupt. Do not edit comment generated here */
    /* Software reset */
    SYSTEM.PRCR.WORD = 0xA502;
    SYSTEM.SWRR = 0xA501;
    /* End user code. Do not edit comment generated here */
}
```

4.2.38 ウォッチドッグタイマ

以下に、コード生成ツールがウォッチドッグタイマ（WDT/IWDT）用として出力する API 関数の一覧を示します。

表 4.39 ウォッチドッグタイマ用 API 関数

API 関数名	機能概要
R_<Config_WDT>_Create	ウォッチドッグタイマ（WDT/IWDT）を制御するうえで必要となる初期化処理を行います。
R_<Config_WDT>_Restart	ウォッチドッグタイマのカウンタをクリアしたのち、カウント処理を再開します。
R_<Config_WDT>_Create_UserInit	ウォッチドッグタイマ（WDT/IWDT）に関するユーザ独自の初期化処理を行います。
r_<Config_WDT>_wuni_interrupt	マスカブル割り込み / ノンマスカブル割り込み（WUNI）の発生に伴う処理を行います。
r_<Config_WDT>_iwuni_interrupt	マスカブル割り込み / ノンマスカブル割り込み（IWUNI）の発生に伴う処理を行います。
r_<Config_WDT>_nmi_interrupt	ノンマスカブル割り込みの発生に伴う処理を行います。

R_<Config_WDT>_Create

ウォッチドッグタイマ（WDT/IWDT）を制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_<Config_WDT>_Restart

ウォッチドッグタイマのカウンタをクリアしたのち、カウント処理を再開します。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_<Config_WDT>_Create_UserInit

ウォッチドッグタイマ（WDT/IWDT）に関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_WDT>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_WDT>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

r_<Config_WDT>_wuni_interrupt

マスカブル割り込み / ノンマスカブル割り込み（WUNI）の発生に伴う処理を行います。

備考　本 API 関数は、ダウンカウンタがアンダフローまたはリフレッシュエラーが発生したとき、マスカブル割り込み / ノンマスカブル割り込み処理として呼び出されます。

[指定形式]

```
void    r_<Config_WDT>_wuni_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_WDT>_iwuni_interrupt

マスカブル割り込み / ノンマスカブル割り込み（IWUNI）の発生に伴う処理を行います。

備考　本 API 関数は、ダウンカウンタがアンダフローまたはリフレッシュエラーが発生したとき、マスカブル割り込み / ノンマスカブル割り込み処理として呼び出されます。

[指定形式]

```
void    r_<Config_WDT>_iwuni_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_WDT>_nmi_interrupt

ノンマスカブル割り込みの発生に伴う処理を行います。

備考　　備考 本 API 関数は、ダウンカウンタがアンダフローまたはリフレッシュエラーが発生したとき、ノンマスカブル割り込み処理として呼び出されます。

[指定形式]

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

[引数]

なし

[戻り値]

なし

使用例

メインループ毎にリフレッシュし、カウンタがアンダフローした際にはソフトウェアリセット

main.c

```
#include "r_smc_entry.h"
void main(void)
{
    while (1U)
    {
        /* Restarts WDT module */
        R_Config_WDT_Restart();
    }
}
```

Config_WDT_user.c

```
void r_Config_WDT_wuni_interrupt(void)
{
    /* Start user code for r_Config_WDT_wuni_interrupt. Do not edit comment generated here */
    /* Software reset */
    SYSTEM.PRCR.WORD = 0xA502;
    SYSTEM.SWRR = 0xA501;
    /* End user code. Do not edit comment generated here */
}
```

4.2.39 連続スキャンモード DSAD

以下に、コード生成ツールが連続スキャンモード DSAD 用として出力する API 関数の一覧を示します。

表 4.40 連続スキャンモード DSAD 用 API 関数

API 関数名	機能概要
R_<Config_DSAD0>_Create	連続スキャンモード DSAD を制御するうえで必要となる初期化処理を行います。
R_<Config_DSAD0>_Start	A/D 変換を開始します。
R_<Config_DSAD0>_Stop	A/D 変換を終了します。
R_<Config_DSAD0>_Set_SoftwareTrigger	ソフトウエアトリガにより A/D 変換を開始します。
R_<Config_DSAD0>_Get_ValueResult	変換結果を獲得します。
R_<Config_DSAD0>_Set_Chm_DisconnectDetection	断線検出アシストの設定を行います。
R_<Config_DSAD0>_Create_UserInit	連続スキャンモード DSAD に関するユーザ独自の初期化処理を行います。
r_<Config_DSAD0>_adin_interrupt	A/D 変換終了割り込みの発生に伴う処理を行います。
r_<Config_DSAD0>_scanendn_interrupt	スキャン終了割り込みの発生に伴う処理を行います。

R_<Config_DSAD0>_Create

連続スキャンモード DSAD を制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void    R_<Config_DSAD0>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_DSAD0>_Start

A/D 変換のトリガ待ちを開始します。

[指定形式]

void	R_<Config_DSAD0>_Start (void);

[引数]

なし

[戻り値]

なし

R_<Config_DSAD0>_Stop

A/D 変換を終了します。

[指定形式]

```
void        R_<Config_DSAD0>_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_DSAD0>_Set_SoftwareTrigger

ソフトウエアトリガによる A/D 変換を開始します。

[指定形式]

```
void      R_<Config_DSAD0>_Set_SoftwareTrigger ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_DSAD0>_Get_ValueResult

変換結果を獲得します。

[指定形式]

```
void    R_<Config_DSAD0>_Get_ValueResult ( uint32_t * const buffer );
```

[引数]

I/O	引数	説明
O	uint32_t * const *buffer*;	獲得した変換結果を格納する領域へのポインタ

[戻り値]

なし

R_<Config_DSAD0>_Set_Ch*m*_DisconnectDetection

断線検出アシストの設定を行います。

[指定形式]

void R_<Config_DSAD0>_Set_Ch*m*_DisconnectDetection(bool *pos*, bool *neg*);

[引数]

I/O	引数	説明
I	bool *pos*;	＋側入力の断線検出アシスト設定 True : 有効 False : 無効
I	bool *neg*;	－側入力の断線検出アシスト設定 True : 有効 False : 無効

[戻り値]

なし

R_<Config_DSAD0>_Create_UserInit

連続スキャンモード DSAD に関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_DSAD0>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_DSAD0>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

r_<Config_DSAD0>_adi*n*_interrupt

変換終了割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_DSAD0>_adi*n*_interrupt (void);

備考 *n* はユニット番号を意味します。

[引数]

なし

[戻り値]

なし

r_<Config_DSAD0>_scanend*n*_interrupt

スキャン完了割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_DSAD0>_scanend*n*_interrupt (void);

備考 *n* はユニット番号を意味します。

[引数]

なし

[戻り値]

なし

使用例

スキャン完了割り込み時に、AD 変換結果を取得する

main.c

```
#include "r_smc_entry.h"
void main(void)
{
    /* Start the DSAD0 converter */
    R_Config_DSAD0_Start();

    while (1U)
    {
        nop();
    }
}
```

Config_DSAD0_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint32_t g_dsad0_value;
/* End user code. Do not edit comment generated here */

void r_Config_DSAD0_scanend0_interrupt(void)
{
    /* Start user code for r_Config_DSAD0_scanend0_interrupt. Do not edit comment generated here */
    /* Get result from the DSAD0 converter */
    R_Config_DSAD0_Get_ValueResult((uint32_t *)&g_dsad0_value);
    /* End user code. Do not edit comment generated here */
}
```

4.2.40 シングルスキャンモード DSAD

以下に、コード生成ツールがシングルスキャンモード DSAD 用として出力する API 関数の一覧を示します。

表 4.41 シングルスキャンモード DSAD 用 API 関数

API 関数名	機能概要
R_<Config_DSAD0>_Create	シングルスキャンモード S12AD を制御するうえで必要となる初期化処理を行います。
R_<Config_DSAD0>_Start	A/D 変換を開始します。
R_<Config_DSAD0>_Stop	A/D 変換を終了します。
R_<Config_DSAD0>_Set_SoftwareTrigger	ソフトウエアトリガにより A/D 変換を開始します。
R_<Config_DSAD0>_Get_ValueResult	変換結果を獲得します。
R_<Config_DSAD0>_Set_Chm_DisconnectDetection	断線検出アシストの設定を行います。
R_<Config_DSAD0>_Create_UserInit	シングルスキャンモード S12AD に関するユーザ独自の初期化処理を行います。
r_<Config_DSAD0>_adin_interrupt	変換終了割り込みの発生に伴う処理を行います。
r_<Config_DSAD0>_scanendn_interrupt	スキャン終了割り込みの発生に伴う処理を行います。

R_<Config_DSAD0>_Create

シングルスキャンモード DSAD を制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void    R_<Config_DSAD0>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_DSAD0>_Start

A/D 変換のトリガ待ちを開始します。

[指定形式]

void	R_<Config_DSAD0>_Start (void);

[引数]

なし

[戻り値]

なし

R_<Config_DSAD0>_Stop

A/D 変換を終了します。

[指定形式]

void	R_<Config_DSAD0>_Stop (void);

[引数]

なし

[戻り値]

なし

R_<Config_DSAD0>_Set_SoftwareTrigger

ソフトウエアトリガによる A/D 変換を開始します。

[指定形式]

```
void    R_<Config_DSAD0>_Set_SoftwareTrigger ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_DSAD0>_Get_ValueResult

変換結果を獲得します。

[指定形式]

```
void      R_<Config_S12AD0>_Get_ValueResult ( uint32_t * const buffer );
```

[引数]

I/O	引数	説明
O	uint32_t * const *buffer*;	獲得した変換結果を格納する領域へのポインタ

[戻り値]

なし

R_<Config_DSAD0>_Set_Ch*m*_DisconnectDetection

断線検出アシストの設定を行います。

[指定形式]

void R_<Config_DSAD0>_Set_Ch*m*_DisconnectDetection (bool *pos*, bool *neg*);

[引数]

I/O	引数	説明
I	bool *pos*;	＋側入力の断線検出アシスト設定 True : 有効 False : 無効
I	bool *neg*;	－側入力の断線検出アシスト設定 True : 有効 False : 無効

[戻り値]

なし

R_<Config_DSAD0>_Create_UserInit

シングルスキャンモード DSAD に関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_DSAD0>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

void	R_<Config_DSAD0>_Create_UserInit (void);

[引数]

なし

[戻り値]

なし

r_<Config_DSAD0>_adi*n*_interrupt

変換終了割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_DSAD0>_adi*n*_interrupt (void);

備考 *n* はユニット番号を意味します。

[引数]

なし

[戻り値]

なし

r_<Config_DSAD0>_scanend*n*_interrupt

スキャン完了割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_DSAD0>_scanend*n*_interruptB (void);

備考 *n* はユニット番号を意味します。

[引数]

なし

[戻り値]

なし

使用例

スキャン完了割り込みで AD 変換結果を取得する

main.c

```
#include "r_smc_entry.h"
void main(void)
{
    /* Start the DSAD0 converter */
    R_Config_DSAD0_Start();

    while (1U)
    {
        nop();
    }
}
```

Config_DSAD0_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint32_t g_dsad0_value;
/* End user code. Do not edit comment generated here */

void r_Config_DSAD0_scanend0_interrupt(void)
{
    /* Start user code for r_Config_DSAD0_scanend0_interrupt. Do not edit comment generated here */
    /* Get result from the DSAD0 converter */
    R_Config_DSAD0_Get_ValueResult((uint32_t *)&g_dsad0_value);
    /* End user code. Do not edit comment generated here */
}
```

4.2.41 Δ-Σモジュールインタフェース

以下に、コード生成ツールがΔ-Σモジュールインタフェース用として出力する API 関数の一覧を示します。

表 4.42 Δ-Σモジュールインタフェース用 API 関数

API 関数名	機能概要
R_<Config_DSMIF0>_Create	Δ-Σモジュールインタフェースを制御するうえで必要となる初期化処理を行います。
R_<Config_DSMIF0>_Start	変換を開始します。
R_<Config_DSMIF0>_Stop	変換を終了します。
R_<Config_DSMIF0>_Create_UserInit	Δ-Σモジュールインタフェースに関するユーザ独自の初期化処理を行います。
r_<Config_DSMIF0>_ocdin_interrupt	過電流検出割り込みの発生に伴う処理を行います。
r_<Config_DSMIF0>_scdin_interrupt	短絡検出割り込みの発生に伴う処理を行います。
r_<Config_DSMIF0>_sumein_interrupt	合計電流エラー割り込みの発生に伴う処理を行います。

R_<Config_DSMIF0>_Create

Δ-Σモジュールインタフェースを制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void    R_<Config_DSMIF0>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_DSMIF0>_Start

変換を開始します。

[指定形式]

void	R_<Config_DSMIF0>_Start (void);

[引数]

なし

[戻り値]

なし

R_<Config_DSMIF0>_Stop

変換を終了します。

[指定形式]

```
void        R_<Config_DSMIF0>_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_DSMIF0>_Create_UserInit

Δ-Σモジュールインタフェースに関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_DSMIF0>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void    R_<Config_DSMIF0>_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_DSMIF0>_ocdi*n*_interrupt

過電流検出割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_DSMIF0>_ocdi*n*_interrupt (void);

備考 *n* はユニット番号を意味します。

[引数]

なし

[戻り値]

なし

r_<Config_DSMIF0>_scdi*n*_interrupt

短絡検出割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_DSMIF0>_scdi*n*_interrupt (void);

備考 *n* はユニット番号を意味します。

[引数]

なし

[戻り値]

なし

r_<Config_DSMIF0>_sumei*n*_interrupt

合計電流エラー検出割り込みの発生に伴う処理を行います。

[指定形式]

void	r_<Config_DSMIF0>_sumei*n*_interrupt (void);

備考 *n* はユニット番号を意味します。

[引数]

なし

[戻り値]

なし

使用例

過電流検出時に、変換を終了する

main.c

```
#include "r_smc_entry.h"
void main(void)
{
    /* Start the DSMIF0 filltering */
    R_Config_DSMIF0_Start();

    while (1U)
    {
        nop();
    }
}
```

Config_DSMIF_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint16_t g_s12ad0_ch000_value;
/* End user code. Do not edit comment generated here */

void r_Config_DSMIF0_ocdi0_interrupt(void)
{
    /* Start user code for r_Config_DSMIF0_ocdi0_interrupt. Do not edit comment generated here
*/
    /* Stop the DSMIF0 convert */
    R_Config_DSMIF0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

4.2.42 アナログフロントエンド

以下に、アナログフロントエンド用として出力する API 関数の一覧を示します。

表 4.43 アナログフロントエンド用 API 関数

API 関数名	機能概要
R_<Config_AFE>_Create	アナログフロントエンドを制御するうえで必要となる初期化処理を行います。
R_<Config_AFE>_Create_UserInit	アナログフロントエンドに関するユーザ独自の初期化処理を行います。

R_<Config_AFE>_Create

アナログフロントエンドを制御するうえで必要となる初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_<Config_AFE>_Create_UserInit

アナログフロントエンドに関するユーザ独自の初期化処理を行います。

備考　本 API 関数は、R_<Config_AFE>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

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

[引数]

なし

[戻り値]

なし

4.2.43 モータ

以下に、モータ用として出力する API 関数の一覧を示します。

表 4.44 モータ用 API 関数

API 関数名	機能概要
R_<Config_MTU3_MTU4>_Create	モータ制御で使用する MTU (相補 PWM モード)および S12AD (シングルスキャンモード)の初期化処理を行います。
R_<Config_MTU3_MTU4>_StartTimerCount	タイマのカウントを開始します。
R_<Config_MTU3_MTU4>_StopTimerCount	タイマのカウントを停止します。
R_<Config_MTU3_MTU4>_StartTimerCtrl	タイマのパルス出力を開始します。
R_<Config_MTU3_MTU4>_StopTimerCtrl	タイマのパルス出力を停止します。
R_<Config_MTU3_MTU4>_UpdDuty	デューティレジスタのバッファを更新します。
R_<Config_MTU3_MTU4>_StartAD	A/D 変換器の動作を開始します。
R_<Config_MTU3_MTU4>_StopAD	A/D 変換器の動作を停止します。
R_<Config_MTU3_MTU4>_AdcGetConvVal	A/D 変換結果を取得します。
R_<Config_MTU3_MTU4>_Create_UserInit	モータ設定時のユーザ独自の初期化処理を行います。
R_<Config_MTU3_MTU4>_CrestInterrupt	山割り込みの発生に伴う処理を行います。
r_<Config_MTU3_MTU4>_ad_interrupt	A/D 変換終了割り込みの発生に伴う処理を行います。

R_<Config_MTU3_MTU4>_Create

モータ制御で使用する MTU (相補 PWM モード)および S12AD (シングルスキャンモード)の初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

```
void    R_<Config_MTU3_MTU4>_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_MTU3_MTU4>_StartTimerCount

タイマのカウントを開始します。

[指定形式]

```
void        R_<Config_MTU3_MTU4>_StartTimerCount ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_MTU3_MTU4>_StopTimerCount

タイマのカウントを停止します。

[指定形式]

```
void        R_<Config_MTU3_MTU4>_StopTimerCount ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_MTU3_MTU4>_StartTimerCtrl

タイマのパルス出力を開始します。

[指定形式]

```
void        R_<Config_MTU3_MTU4>_StartTimerCtrl ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_MTU3_MTU4>_StopTimerCtrl

タイマのパルス出力を停止します。

[指定形式]

void	R_<Config_MTU3_MTU4>_StopTimerCtrl (void);

[引数]

なし

[戻り値]

なし

R_<Config_MTU3_MTU4>_UpdDuty

デューティレジスタのバッファを更新します。

[指定形式]

3 相ブラシレス DC モータ

```
void R_<Config_MTU3_MTU4>_UpdDuty (uint16_t duty_u, uint16_t duty_v, uint16_t duty_w);
```

2 相ステッピングモータ

```
void R_<Config_MTU3_MTU4>_UpdDuty (uint16_t duty_a, uint16_t duty_b);
```

[引数]

3 相ブラシレス DC モータ

I/O	引数	説明
O	uint16_t duty_u	U phase duty register value
O	uint16_t duty_v	V phase duty register value
O	uint16_t duty_w	W phase duty register value

2 相ステッピングモータ

I/O	引数	説明
O	uint16_t duty_a	A phase duty register value
O	uint16_t duty_b	B phase duty register value

[戻り値]

なし

R_<Config_MTU3_MTU4>_StartAD

A/D 変換器の動作を開始します。

[指定形式]

void	R_<Config_MTU3_MTU4>_StartAD (void);

[引数]

なし

[戻り値]

なし

R_<Config_MTU3_MTU4>_StopAD

A/D 変換器の動作を停止します。

[指定形式]

```
void        R_<Config_MTU3_MTU4>_StopAD ( void );
```

[引数]

なし

[戻り値]

なし

R_<Config_MTU3_MTU4>_AdcGetConvVal

A/D 変換結果を取得します。

[指定形式]

3 相ブラシレス DC モータ

```
void        R_<Config_MTU3_MTU4>_AdcGetConvVal ( r_mtr_adc_tb *mtr_ad_data );
```

2 相ステッピングモータ

```
void        R_<Config_MTU3_MTU4>_AdcGetConvVal ( r_mtr_adc_ts *mtr_ad_data );
```

[引数]

3-Phase Brushless DC Motor

I/O	引数	説明
O	r_mtr_adc_tb *mtr_ad_data 構造体定義: typedef struct { uint16_t u2_iu_ad; uint16_t u2_iv_ad; uint16_t u2_iw_ad; uint16_t u2_vdc_ad; uint16_t u2_vphase_u_ad; uint16_t u2_vphase_v_ad; uint16_t u2_vphase_w_ad; } r_mtr_adc_tb;	変換結果を格納する領域へのポインタ

2 相ステッピングモータ

I/O	引数	説明
O	r_mtr_adc_ts *mtr_ad_data 構造体定義: typedef struct { uint16_t u2_ia_ad; uint16_t u2_ib_ad; uint16_t u2_vdc_ad; uint16_t u2_vphase_a_ad; uint16_t u2_vphase_b_ad; } r_mtr_adc_ts;	変換結果を格納する領域へのポインタ

[戻り値]

なし

R_<Config_MTU3_MTU4>_Create_UserInit

モータ設定時のユーザ独自の初期化処理を行います。

備考　本関数は、R_<Config_MTU3_MTU4>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

```
void    R_<Config_MTU3_MTU4>_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_MTU3_MTU4>_CrestInterrupt

山割り込みの発生に伴う処理を行います。

備考　本関数は、タイマカウンタ(TCNT)の値とタイマジェネラルレジスタ(TGRA3 または TGRA6）の値が一致した場合に発生するコンペアマッチ割り込みに対応した割り込み処理として呼び出されます。

[指定形式]

```
void    r_<Config_MTU3_MTU4>_CrestInterrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_<Config_MTU3_MTU4>_ad_interrupt

A/D 変換終了割り込みの発生に伴う処理を行います。

備考　GUI 上で選択された S12AD ユニットのうち、最小のユニット番号の 1 ユニットの割り込みが有効となります。

[指定形式]

```
void    r_<Config_MTU3_MTU4>_ad_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

4.2.44 LCD コントローラ

以下に、コード生成ツールが LCD コントローラ用として出力する API 関数の一覧を示します。

表 4.45 LCD コントローラ用 API 関数

API 関数名	機能概要
R_<Config_LCD>_Create	LCD コントローラの初期化処理を行います。
R_<Config_LCD>_Start	LCD 表示を開始します。
R_<Config_LCD>_Stop	LCD 表示を終了します。
R_<Config_LCD>_Voltage_On	昇圧回路または容量分割回路を有効にします。
R_<Config_LCD>_Voltage_Off	昇圧回路または容量分割回路を無効にします。
R_<Config_LCD>_Create_UserInit	LCD コントローラ設定時のユーザ独自の初期化処理を行います。

R_<Config_LCD>_Create

LCD コントローラの初期化処理を行います。

備考　本 API 関数は、main()関数を実行する前に、R_Systeminit から呼び出されます。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_<Config_LCD>_Start

LCD 表示を開始します。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_<Config_LCD>_Stop

LCD 表示を終了します。

[指定形式]

void	R_<Config_LCD>_Stop (void);

[引数]

なし

[戻り値]

なし

R_<Config_LCD>_Voltage_On

昇圧回路または容量分割回路を有効にします。

[指定形式]

void	R_<Config_LCD>_Voltage_On (void);

[引数]

なし

[戻り値]

なし

R_<Config_LCD>_Voltage_Off

昇圧回路または容量分割回路を無効にします。

[指定形式]

void	R_<Config_LCD>_Voltage_Off (void);

[引数]

なし

[戻り値]

なし

R_<Config_LCD>_Create_UserInit

LCD コントローラ設定時のユーザ独自の初期化処理を行います。

備考　本関数は、R_<Config_LCD>_Create のコールバック・ルーチンとして呼び出されます。

[指定形式]

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

[引数]

なし

[戻り値]

なし

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2018.07.01	―	初版発行
1.01	2018.07.10	19	備考を追記
1.02	2019.11.05	19	初期化を追記
		483 -511	新コンポーネントを追加 ● 連続スキャンモード DSAD ● シングルスキャンモード DSAD ● Δ-Σモジュールインタフェース
1.03	2020.01.20	11, 13	出力ファイル一覧に API 関数を追加 ● ソフトウェアカウンタクリア ● プログラマブルゲインアンプ動作条件設定
1.04	2020.07.20	12, 200 - 232	出力ファイル一覧および I2C マスタモードの構成を変更 ● RIIC と SCI 簡易 I2C モードの API を分離
		207	備考を修正
		18, 502	API 関数を追加 ● 断線検出アシスト設定
		527 - 529	新コンポーネントを追加 ● アナログフロントエンド
1.05	2020.10.20	530 – 542	新コンポーネントを追加 ● モータ
1.06	2022.02.16	137 - 142	U 相、V 相、W 相の Start API と Stop API を分割
		219 248	API 名とパラメータ概要を更新
		299 300	新しい API を追加 ● MTU チャンネルのカウンタを同時に Start/Stop
		455 – 460	送受信バッファパラメータのデータ型の記述を変更
		476 477	受信バッファパラメータのデータ型の記述を変更
1.07	2022.08.01	122 - 125	DOC コンポーネントを更新 ● 3 つのパラメータで定義されている R_<Config_DOC>SetMode API を追加 ● “uint32_t” を値の type に追加
		206	シンプル I2C モードの RSCI サポート情報を追加
		375 - 418	● SCI/SCIF 調歩同期式モードコンポーネントおよび SCI/SCIF クロック同期式モードコンポーネントの RSCI サポート情報を追加 ● SCI/SCIF 調歩同期式モードコンポーネントおよび SCI/SCIF クロック同期式モードコンポーネントの r_<Config_SCI0>_receive_interrupt API の記述を更新
		437	スマートカードインタフェースコンポーネントの RSCI サポート情報を追加

Rev.	発行日	改訂内容	
		ページ	ポイント
1.07	2022.08.01	452 - 472	● 簡易 SPI モードの RSCI サポート情報を追加 ● 新しい API を追加（受信エラー割り込みの発生に伴う処理を行う） ● SPI クロック同期式モードコンポーネントの r_<Config_SCI0>_receive_interrupt API の記述を更新
		551 - 557	新しいコンポーネントを追加 ● LCD コントローラ
1.08	2023.10.20	15, 334, 354, 355	新しい API を追加 ● アラーム割り込みの許可/禁止
		151	R_<config_DMAC0>_Set_SoftwareTrigger に備考を追加
		173, 174	R_<config_GPT0>_Start と R_<Config_GPT0>_Stop に備考と追加

スマート・コンフィグレータ ユーザーズマニュアル
RX APIリファレンス編

発行年月日　2023年10月20日　　Rev.1.08

発行　　ルネサス エレクトロニクス株式会社
〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

スマート・コンフィグレータ

ルネサス エレクトロニクス株式会社

R20UT4360JJ0108