User's Manual

CS+ コード生成ツール

統合開発環境

ユーザーズマニュアル　RX API リファレンス編

対象デバイス

RX ファミリ

対象ツール

CS+ V4.01.00

ルネサス エレクトロニクス
www.renesas.com

Rev.1.30　2016.10

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して、お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
2. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
3. 本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害に関し、当社は、何らの責任を負うものではありません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、
   各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、
   家電、工作機械、パーソナル機器、産業用ロボット等

   高品質水準：　輸送機器（自動車、電車、船舶等）、交通用信号機器、
   防災・防犯装置、各種安全装置等

   当社製品は、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。たとえ、意図しない用途に当社製品を使用したことによりお客様または第三者に損害が生じても、当社は一切その責任を負いません。なお、ご不明点がある場合は、当社営業にお問い合わせください。
6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 本資料に記載されている当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。また、当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途に使用しないでください。当社製品または技術を輸出する場合は、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。
10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注１において定義された当社の開発、製造製品をいいます。

(2012. 4)

このマニュアルの使い方

このマニュアルは，RX ファミリ用アプリケーション・システムを開発する際の統合開発環境である CS+ について説明します。
CS+ は，RX ファミリの統合開発環境（ソフトウエア開発における，設計，実装，デバッグなどの各開発フェーズに必要なツールをプラットフォームである IDE に統合）です。統合することで，さまざまなツールを使い分ける必要がなく，本製品のみを使用して開発のすべてを行うことができます。

対象者	このマニュアルは，コード生成ツールの機能を理解し，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，コード生成ツール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。 1. 概　　説 2. 出力ファイル 3.API 関数
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 . . . 3
1.1 概　要 . . . 3
1.2 特　長 . . . 3
2. 出力ファイル . . . 4
2.1 説　明 . . . 4
3. API 関数 . . . 14
3.1 概　要 . . . 14
3.2 関数リファレンス . . . 14
3.2.1 共　通 . . . 16
3.2.2 クロック発生回路 . . . 31
3.2.3 電圧検出回路（LVD） . . . 37
3.2.4 クロック周波数精度測定回路（CAC） . . . 43
3.2.5 消費電力低減機能 . . . 51
3.2.6 割り込みコントローラ（ICU） . . . 61
3.2.7 バ　ス . . . 76
3.2.8 DMA コントローラ（DMAC） . . . 83
3.2.9 データ・トランスファ・コントローラ（DTC） . . . 93
3.2.10 イベント・リンク・コントローラ（ELC） . . . 98
3.2.11 I/O ポート . . . 107
3.2.12 マルチファンクション・タイマ・パルス・ユニット 2（MTU2） . . . 110
3.2.13 マルチファンクション・タイマ・パルス・ユニット 3（MTU3） . . . 120
3.2.14 ポート・アウトプット・イネーブル 2（POE2） . . . 130
3.2.15 ポート・アウトプット・イネーブル 3（POE3） . . . 138
3.2.16 汎用 PWM タイマ（GPT） . . . 148
3.2.17 16 ビットタイマ・パルス・ユニット（TPU） . . . 161
3.2.18 8 ビットタイマ・パルス・ユニット（TMR） . . . 169
3.2.19 プログラマブル・パルスジェネレータ（PPG） . . . 176
3.2.20 コンペア・マッチ・タイマ（CMT） . . . 179
3.2.21 コンペア・マッチ・タイマ W（CMTW） . . . 185
3.2.22 リアルタイム・クロック（RTC） . . . 193
3.2.23 ウォッチドッグ・タイマ（WDT） . . . 216
3.2.24 独立ウォッチドッグ・タイマ（IWDT） . . . 222
3.2.25 シリアル・コミュニケーション・インタフェース（SCI） . . . 228
3.2.26 FIFO 内蔵シリアル・コミュニケーション・インターフェース（SCIFA） . . . 255
3.2.27 I^2C バス・インタフェース（RIIC） . . . 272
3.2.28 シリアル・ペリフェラル・インタフェース（RSPI） . . . 290

3.2.29 CRC 演算器（CRC）. 304
3.2.30 12 ビット A/D コンバータ（S12AD）. 312
3.2.31 D/A コンバータ（DA）. 322
3.2.32 12 ビット D/A コンバータ（R12DA）. 328
3.2.33 コンパレータ B（CMPB）. 336
3.2.34 データ演算回路（DOC）. 342
3.2.35 ローパワータイマ（LPT）. 350
3.2.36 コンパレータ C（CMPC）. 355
3.2.37 LCD コントローラ / ドライバ（LCD）. 361

改訂記録 368

1. 概　　説

CS+ は，アプリケーション・システムを開発する際の統合開発環境であり，設計／コーディング／ビルド／デバッグなどといった一連の作業を実施することができます。
本章では，設計ツール（コード生成）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，コード生成ツールの特長を示します。

- コード生成機能
  コード生成では，GUI ベースで設定した情報に応じたデバイス・ドライバ・プログラムを出力するだけでなく，main 関数を含んだサンプル・プログラムなどといったビルド環境一式を出力することもできます。

- レポート機能
  コード生成を用いて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
  コード生成が出力するファイル名，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。

- 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

周辺機能	ファイル名	API 関数名	出力 (*1)
共　　通	r_cg_main.c	main R_MAIN_UserInit	○ ○
	r_dbsct.c	—	—
	r_cg_intprg.c	r_undefined_exception r_privileged_exception r_floatingpoint_exception r_access_exception r_nmi_exception r_brk_exception r_reserved_exception r_icu_group_n_interrupt	○ ○ ○ ○ ○ ○ ○ ○
	r_cg_resetprg.c	PowerON_Reset PowerON_Reset_PC	○ ○
	r_cg_sbrk.c	—	—
	r_cg_vecttbl.c	—	—
	r_cg_sbrk.h	—	—
	r_cg_stacksct.h	—	—
	r_cg_vect.h	—	—
	r_cg_hardware_setup.c	HardwareSetup R_Systeminit	○ ○
	r_cg_macrodriver.h	—	—
	r_cg_userdefine.h	—	—
クロック発生回路	r_cg_cgc.c	R_CGC_Create R_CGC_Set_ClockMode	○ ×
	r_cg_cgc_user.c	R_CGC_Create_UserInit r_cgc_oscillation_stop_nmi_interrupt r_cgc_oscillation_stop_interrupt	× ○ ○
	r_cg_cgc.h	—	—
電圧検出回路（LVD）	r_cg_lvd.c	R_LVDn_Create R_LVDn_Start R_LVDn_Stop	○ ○ ○
	r_cg_lvd_user.c	R_LVDn_Create_UserInit r_lvd_lvdn_interrupt	× ○
	r_cg_lvd.h	—	—

周辺機能	ファイル名	API 関数名	出力 (*1)
クロック周波数精度測定回路（CAC）	r_cg_cac.c	R_CAC_Create R_CAC_Start R_CAC_Stop	○ ○ ○
	r_cg_cac_user.c	R_CAC_Create_UserInit r_cac_mendf_interrupt r_cac_ferrf_interrupt r_cac_ovff_interrupt	× ○ ○ ○
	r_cg_cac.h	–	–
消費電力低減機能	r_cg_lpc.c	R_LPC_Create R_LPC_AllModuleClockStop R_LPC_ChangeSleepModeReturnClock R_LPC_Sleep R_LPC_DeepSleep R_LPC_DeepSoftwareStandby R_LPC_SoftwareStandby R_LPC_ChangeOperatingPowerControl	○ ○ ○ ○ ○ ○ ○ ○
	r_cg_lpc_user.c	R_LPC_Create_UserInit	×
	r_cg_lpc.h	–	–
割り込みコントローラ（ICU）	r_cg_icu.c	R_ICU_Create R_ICU_IRQn_Start R_ICU_IRQn_Stop R_ICU_Software_Start R_ICU_Software2_Start R_ICU_Software_Stop R_ICU_Software2_Stop R_ICU_SoftwareInterrupt_Generate R_ICU_SoftwareInterrupt2_Generate	○ ○ ○ ○ ○ ○ ○ ○ ○
	r_cg_icu_user.c	R_ICU_Create_UserInit r_icu_irqn_interrupt r_icu_software_interrupt r_icu_software2_interrupt r_icu_nmi_interrupt	× ○ ○ ○ ○
	r_cg_icu.h	–	–
バ　　ス	r_cg_bsc.c	R_BSC_Create R_BSC_Error_Monitoring_Start R_BSC_Error_Monitoring_Stop R_BSC_InitializeSDRAM	○ ○ ○ ○
	r_cg_bsc_user.c	R_BSC_Create_UserInit r_bsc_buserr_interrupt	× ○
	r_cg_bsc.h	–	–

周辺機能	ファイル名	API 関数名	出力 (*1)
DMA コントローラ（DMAC）	r_cg_dmac.c	R_DMAC_Create R_DMACn_Start R_DMACn_Stop R_DMACn_Set_SoftwareTrigger R_DMACn_Clear_SoftwareTrigger	○ ○ ○ ○ ○
	r_cg_dmac_user.c	r_dmac_dmacni_interrupt r_dmacn_callback_transfer_end r_dmacn_callback_transfer_escape_end R_DMAC_Create_UserInit	○ ○ ○ ×
	r_cg_dmac.h	–	–
データ・トランスファ・コントローラ（DTC）	r_cg_dtc.c	R_DTC_Create R_DTCm_Start R_DTCm_Stop	○ ○ ○
	r_cg_dtc_user.c	R_DTC_Create_UserInit	×
	r_cg_dtc.h	–	–
イベント・リンク・コントローラ（ELC）	r_cg_elc.c	R_ELC_Create R_ELC_Start R_ELC_Stop R_ELC_GenerateSoftwareEvent R_ELC_Set_PortBuffern R_ELC_Get_PortBuffern	○ ○ ○ ○ ○ ○
	r_cg_elc_user.c	R_ELC_Create_UserInit r_elc_elsrni_interrupt	× ○
	r_cg_elc.h	–	–
I/O ポート	r_cg_port.c	R_PORT_Create	○
	r_cg_port_user.c	R_PORT_Create_UserInit	×
	r_cg_port.h	–	–
マルチファンクション・タイマ・パルス・ユニット 2（MTU2）	r_cg_mtu2.c	R_MTU2_Create R_MTU2_Cn_Start R_MTU2_Cn_Stop	○ ○ ○
	r_cg_mtu2_user.c	R_MTU2_Create_UserInit r_mtu2_tgimn_interrupt r_mtu2_cj_tgimn_interrupt r_mtu2_tcivn_interrupt r_mtu2_cj_tcivn_interrupt r_mtu2_tciun_interrupt	× ○ ○ ○ ○ ○
	r_cg_mtu2.h	–	–

周辺機能	ファイル名	API 関数名	出力 (*1)
マルチファンクション・タイマ・パルス・ユニット 3（MTU3）	r_cg_mtu3.c	R_MTU3_Create R_MTU3_Cn_Start R_MTU3_Cn_Stop	○ ○ ○
	r_cg_mtu3_user.c	R_MTU3_Create_UserInit r_mtu3_tgimn_interrupt r_mtu3_cj_tgimn_interrupt r_mtu3_tcivn_interrupt r_mtu3_cj_tcivn_interrupt r_mtu3_tciun_interrupt	× ○ ○ ○ ○ ○
	r_cg_mtu3.h	–	–
ポート・アウトプット・イネーブル 2（POE2）	r_cg_poe2.c	R_POE2_Create R_POE2_Start R_POE2_Stop R_POE2_Set_HiZ_MTUn R_POE2_Clear_HiZ_MTUn	○ ○ ○ ○ ○
	r_cg_poe2_user.c	R_POE2_Create_UserInit r_poe2_oein_interrupt	× ○
	r_cg_poe2.h	–	–
ポート・アウトプット・イネーブル 3（POE3）	r_cg_poe3.c	R_POE3_Create R_POE3_Start R_POE3_Stop R_POE3_Set_HiZ_MTUn R_POE3_Clear_HiZ_MTUn R_POE3_Set_HiZ_GPTn R_POE3_Clear_HiZ_GPTn	○ ○ ○ ○ ○ ○ ○
	r_cg_poe3_user.c	R_POE3_Create_UserInit r_poe3_oein_interrupt	× ○
	r_cg_poe3.h	–	–
汎用 PWM タイマ（GPT）	r_cg_gpt.c	R_GPT_Create R_GPTn_Start R_GPTn_Stop R_GPTn_HardwareStart R_GPTn_HardwareStop	○ ○ ○ ○ ○
	r_cg_gpt_user.c	R_GPT_Create_UserInit r_gpt_gtcimn_interrupt r_gpt_gtcivn_interrupt r_gpt_gtciun_interrupt r_gpt_gdten_interrupt r_gpt_etgip_interrupt r_gpt_etgin_interrupt	× ○ ○ ○ ○ ○ ○
	r_cg_gpt.h	–	–

周辺機能	ファイル名	API 関数名	出力 (*1)
16 ビットタイマ・パルス・ユニット（TPU）	r_cg_tpu.c	R_TPU_Create R_TPUn_Start R_TPUn_Stop	○ ○ ○
	r_cg_tpu_user.c	R_TPU_Create_UserInit r_tpu_tginm_interrupt r_tpu_tcinv_interrupt r_tpu_tcinu_interrupt	× ○ ○ ○
	r_cg_tpu.h	–	–
8 ビットタイマ・パルス・ユニット（TMR）	r_cg_tmr.c	R_TMR_Create R_TMRn_Start R_TMRn_Stop	○ ○ ○
	r_cg_tmr_user.c	R_TMR_Create_UserInit r_tmr_cmimn_interrupt r_tmr_ovin_interrupt	× ○ ○
	r_cg_tmr.h	–	–
プログラマブル・パルスジェネレータ（PPG）	r_cg_ppg.c	R_PPG_Create	○
	r_cg_ppg_user.c	R_PPG_Create_UserInit	×
	r_cg_ppg.h	–	–
コンペア・マッチ・タイマ（CMT）	r_cg_cmt.c	R_CMTn_Create R_CMTn_Start R_CMTn_Stop	○ ○ ○
	r_cg_cmt_user.c	R_CMTn_Create_UserInit r_cmt_cmin_interrupt	× ○
	r_cg_cmt.h	–	–
コンペア・マッチ・タイマ W（CMTW）	r_cg_cmtw.c	R_CMTWn_Create R_CMTWn_Start R_CMTWn_Stop	○ ○ ○
	r_cg_cmtw_user.c	R_CMTWn_Create_UserInit r_cmtw_cmwin_interrupt r_cmtw_icmin_interrupt r_cmtw_ocmin_interrupt	× ○ ○ ○
	r_cg_cmtw.h	–	–

周辺機能	ファイル名	API 関数名	出力 (*1)
リアルタイム・クロック（RTC）	r_cg_rtc.c	R_RTC_Create	○
		R_RTC_Set_CalendarAlarm	○
		R_RTC_Set_BinaryAlarm	○
		R_RTC_Set_ConstPeriodInterruptOn	○
		R_RTC_Set_ConstPeriodInterruptOff	○
		R_RTC_Set_CarryInterruptOn	○
		R_RTC_Set_CarryInterruptOff	○
		R_RTC_Set_RTCOUTOn	○
		R_RTC_Set_RTCOUTOff	○
		R_RTC_Start	○
		R_RTC_Stop	○
		R_RTC_Restart	○
		R_RTC_Set_CalendarCounterValue	○
		R_RTC_Get_CalendarCounterValue	○
		R_RTC_Set_BinaryCounterValue	○
		R_RTC_Get_BinaryCounterValue	○
		R_RTC_Get_CalendarTimeCaptureValuen	○
		R_RTC_Get_BinaryTimeCaptureValuen	○
	r_cg_rtc_user.c	R_RTC_Create_UserInit	×
		r_rtc_alm_interrupt	○
		r_rtc_prd_interrupt	○
		r_rtc_cup_interrupt	○
	r_cg_rtc.h	—	—
ウォッチドッグ・タイマ（WDT）	r_cg_wdt.c	R_WDT_Create	○
		R_WDT_Restart	○
	r_cg_wdt_user.c	R_WDT_Create_UserInit	×
		r_wdt_nmi_interrupt	○
		r_wdt_wuni_interrupt	○
	r_cg_wdt.h	—	—
独立ウォッチドッグ・タイマ（IWDT）	r_cg_iwdt.c	R_IWDT_Create	○
		R_IWDT_Restart	○
	r_cg_iwdt_user.c	R_IWDT_Create_UserInit	×
		r_iwdt_nmi_interrupt	○
		r_iwdt_iwuni_interrupt	○
	r_cg_iwdt.h	—	—

周辺機能	ファイル名	API 関数名	出力 (*1)
シリアル・コミュニケーション・インタフェース（SCI）	r_cg_sci.c	R_SCIn_Create	○
		R_SCIn_Start	○
		R_SCIn_Stop	○
		R_SCIn_Serial_Send	○
		R_SCIn_Serial_Receive	○
		R_SCIn_Serial_Multiprocessor_Send	○
		R_SCIn_Serial_Multiprocessor_Receive	○
		R_SCIn_Serial_Send_Receive	○
		R_SCIn_SmartCard_Send	○
		R_SCIn_SmartCard_Receive	○
		R_SCIn_IIC_Master_Send	○
		R_SCIn_IIC_Master_Receive	○
		R_SCIn_SPI_Master_Send	○
		R_SCIn_SPI_Master_Send_Receive	○
		R_SCIn_SPI_Slave_Send	○
		R_SCIn_SPI_Slave_Send_Receive	○
		R_SCIn_IIC_StartCondition	○
		R_SCIn_IIC_StopCondition	○
	r_cg_sci_user.c	R_SCIn_Create_UserInit	×
		r_scin_transmitend_interrupt	○
		r_scin_transmit_interrupt	○
		r_scin_receive_interrupt	○
		r_scin_receiveerror_interrupt	○
		r_scin_callback_transmitend	○
		r_scin_callback_receiveend	○
		r_scin_callback_receiveerror	○
	r_cg_sci.h	—	—
FIFO 内蔵シリアル・コミュニケーション・インターフェース（SCIFA）	r_cg_scifa.c	R_SCIFAn_Create	○
		R_SCIFAn_Start	○
		R_SCIFAn_Stop	○
		R_SCIFAn_Serial_Send	○
		R_SCIFAn_Serial_Receive	○
		R_SCIFAn_Serial_Send_Receive	○
	r_cg_scifa_user.c	R_SCIFAn_Create_UserInit	×
		r_scifan_teif_interrupt	○
		r_scifan_txif_interrupt	○
		r_scifan_rxif_interrupt	○
		r_scifan_erif_interrupt	○
		r_scifan_brif_interrupt	○
		r_scifan_drif_interrupt	○
		r_scifan_callback_transmitend	○
		r_scifan_callback_receiveend	○
		r_scifan_callback_error	○
	r_cg_scifa.h	—	—

周辺機能	ファイル名	API 関数名	出力 (*1)
I²C バス・インタフェース（RIIC）	r_cg_riic.c	R_RIICn_Create R_RIICn_Start R_RIICn_Stop R_RIICn_Master_Send R_RIICn_Master_Receive R_RIICn_Slave_Send R_RIICn_Slave_Receive R_RIICn_StartCondition R_RIICn_StopCondition	○ ○ ○ ○ ○ ○ ○ ○ ○
	r_cg_riic_user.c	R_RIICn_Create_UserInit r_riicn_error_interrupt r_riicn_receive_interrupt r_riicn_transmit_interrupt r_riicn_transmitend_interrupt r_riicn_callback_receiveerror r_riicn_callback_transmitend r_riicn_callback_receiveend	× ○ ○ ○ ○ ○ ○ ○
	r_cg_riic.h	–	–
シリアル・ペリフェラル・インタフェース（RSPI）	r_cg_rspi.c	R_RSPIn_Create R_RSPIn_Start R_RSPIn_Stop R_RSPIn_Send R_RSPIn_Send_Receive	○ ○ ○ ○ ○
	r_cg_rspi_user.c	R_RSPIn_Create_UserInit r_rspin_receive_interrupt r_rspin_transmit_interrupt r_rspin_error_interrupt r_rspin_idle_interrupt r_rspin_callback_receiveend r_rspin_callback_error r_rspin_callback_transmitend	× ○ ○ ○ ○ ○ ○ ○
	r_cg_rspi.h	–	–
CRC 演算器（CRC）	r_cg_crc.c	R_CRC_SetCRC8 R_CRC_SetCRC16 R_CRC_SetCCITT R_CRC_SetCRC32 R_CRC_SetCRC32C R_CRC_Input_Data R_CRC_Get_Result	○ ○ ○ ○ ○ ○ ○
	r_cg_crc.h	–	–
12 ビット A/D コンバータ（S12AD）	r_cg_s12ad.c	R_S12ADn_Create R_S12ADn_Start R_S12ADn_Stop R_S12ADn_Get_ValueResult R_S12ADn_Set_CompareValue	○ ○ ○ ○ ○
	r_cg_s12ad_user.c	R_S12ADn_Create_UserInit r_s12adn_interrupt r_s12adn_groupb_interrupt r_s12adn_compare_interrupt	× ○ ○ ○
	r_cg_s12ad.h	–	–

周辺機能	ファイル名	API 関数名	出力 (*1)
D/A コンバータ（DA）	r_cg_da.c	R_DA_Create R_DAm_Start R_DAm_Stop R_DAm_Set_ConversionValue	○ ○ ○ ○
	r_cg_da_user.c	R_DA_Create_UserInit	×
	r_cg_da.h	–	–
12 ビット D/A コンバータ（R12DA）	r_cg_r12da.c	R_R12DA_Create R_R12DAn_Start R_R12DAn_Stop R_R12DAn_Set_ConversionValue R_R12DA_Sync_Start R_R12DA_Sync_Stop	○ ○ ○ ○ ○ ○
	r_cg_r12da_user.c	R_R12DA_Create_UserInit	×
	r_cg_r12da.h	–	–
コンパレータ B（CMPB）	r_cg_cmpb.c	R_CMPB_Create R_CMPBn_Start R_CMPBn_Stop	○ ○ ○
	r_cg_doc_user.c	R_CMPB_Create_UserInit r_cmpb_cmpbn_interrupt	× ○
	r_cg_doc.h	–	–
データ演算回路（DOC）	r_cg_doc.c	R_DOC_Create R_DOC_SetMode R_DOC_WriteData R_DOC_GetResult R_DOC_ClearFlag	○ ○ ○ ○ ○
	r_cg_doc_user.c	R_DOC_Create_UserInit r_doc_dopcf_interrupt	× ○
	r_cg_doc.h	–	–
ローパワータイマ（LPT）	r_cg_lpt.c	R_LPT_Create R_LPT_Start R_LPT_Stop	○ ○ ○
	r_cg_lpt_user.c	R_LPT_Create_UserInit	×
	r_cg_lpt.h	–	–
コンパレータ C（CMPC）	r_cg_cmpc.c	R_CMPC_Create R_CMPCn_Start R_CMPCn_Stop	○ ○ ○
	r_cg_cmpc_user.c	R_CMPC_Create_UserInit r_cmpc_cmpcn_interrupt	× ○
	r_cg_cmpc.h	-	–

周辺機能	ファイル名	API 関数名	出力 (*1)
LCD コントローラ / ドライバ（LCD）	r_cg_cld.c	R_LCD_Create R_LCD_Start R_LCD_Stop R_LCD_Voltage_On R_LCD_Voltage_Off	○ ○ ○ ○ ○
	r_cg_lcd_user.c	R_LCD_Create_UserInit	×
	r_cg_lcd.h	-	―

*1　［コード生成 . プロパティ ,API 関数の出力設定］がデフォルト（設定にあわせてすべて出力する）の場合
　　○：周辺機能パネルの設定により自動で出力される。
　　×：" コード・プレビュー " から、API のプロパティを開き、" 関数を使用する " の設定により、出力される。

3. 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。

3.1 概　　要

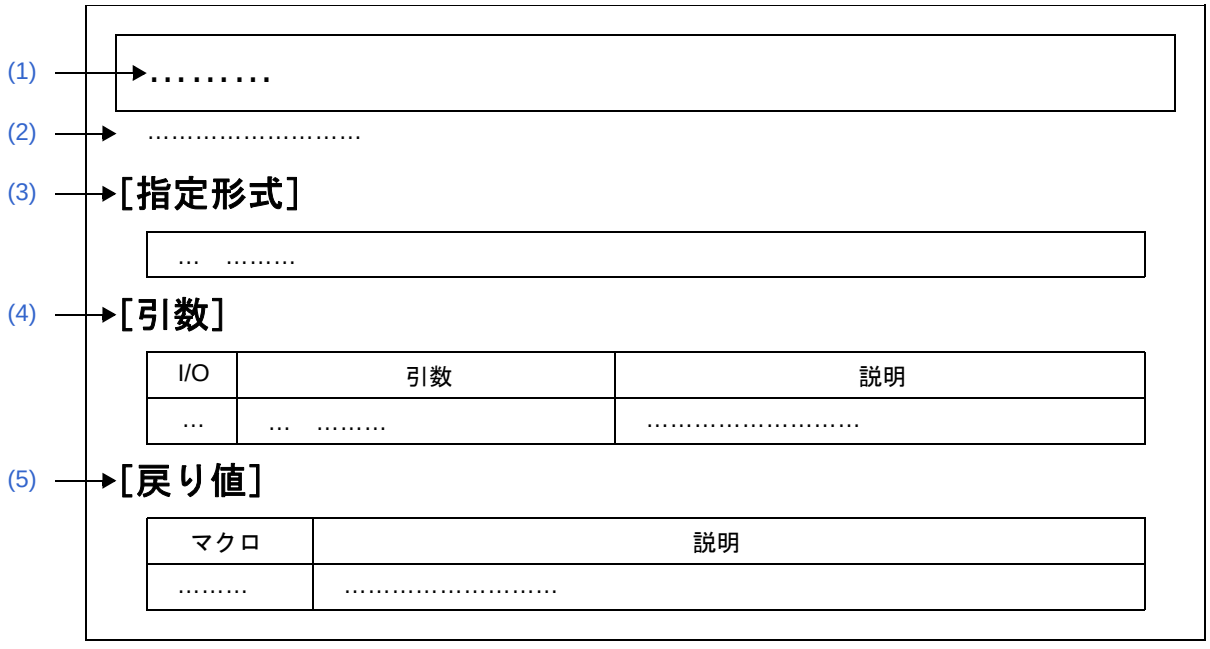
以下に，コード生成が API 関数を出力する際の命名規則を示します。

- 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。

3.2 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 3.1　　API 関数の記述フォーマット



(1)　名称
API 関数の名称を示しています。

(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

3.2.1 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 3.1　　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。
PowerON_Reset_PC	リセットの発生に伴う処理を行います。
r_privileged_exception	特権命令例外の発生に伴う処理を行います。
r_floatingpoint_exception	浮動小数点例外の発生に伴う処理を行います。
r_access_exception	アクセス例外の発生に伴う処理を行います。
r_nmi_exception	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_brk_exception	無条件トラップの発生に伴う処理を行います。
r_reserved_exception	例外（未定義命令例外，リセット，ノンマスカブル割り込み，無条件トラップ以外）の発生に伴う処理を行います。
HardwareSetup	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。
main	main 関数です。
R_MAIN_UserInit	ユーザ独自の初期化処理を行います。
r_icu_group_n_interrupt	グループ割り込み発生時の処理を行います。

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

PowerON_Reset_PC

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_PC ( void );
```

[引数]

なし

[戻り値]

なし

r_privileged_exception

特権命令例外の発生に伴う処理を行います。

備考 本 API 関数は，ユーザモードで特権命令の実行を検出した場合に発生する特権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_privileged_exception ( void );
```

[引数]

なし

[戻り値]

なし

r_floatingpoint_exception

浮動小数点例外の発生に伴う処理を行います。

備考 本 API 関数は，IEEE754 規格で規定された 5 つの例外自称 (オーバフロー , アンダフロー , 精度異常 , ゼロ除算 , 無効演算) の他 , 非実装処理を検出した場合に発生する浮動小数点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_floatingpoint_exception ( void );
```

[引数]

なし

[戻り値]

なし

r_access_exception

アクセス例外の発生に伴う処理を行います。

備考 本 API 関数は，CPU からのメモリアクセスによるエラーが検出された場合に発生するアクセス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_access_exception ( void );
```

[引数]

なし

[戻り値]

なし

r_nmi_exception

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　本 API 関数は，ノンマスカブル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_nmi_exception ( void );
```

[引数]

なし

[戻り値]

なし

r_brk_exception

無条件トラップの発生に伴う処理を行います。

備考 本 API 関数は，INT 命令，または BRK 命令が発行された場合に発生する無条件トラップ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_brk_exception ( void );
```

[引数]

なし

[戻り値]

なし

r_reserved_exception

例外（未定義命令例外，リセット，ノンマスカブル割り込み，無条件トラップ以外）の発生に伴う処理を行います。

備考　本 API 関数は，未定義命令例外，リセット，ノンマスカブル割り込み，無条件トラップ以外の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_reserved_exception ( void );
```

[引数]

なし

[戻り値]

なし

HardwareSetup

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    HardwareSetup ( 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，HardwareSetup 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

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

R_MAIN_UserInit

ユーザ独自の初期化処理を行います。

備考 本 API 関数は，main 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_MAIN_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_icu_group_*n*_interrupt

グループ割り込み発生時の処理を行います。

[指定形式]

```
void    void r_icu_group_n_interrupt ( void );
```

備考 *n* は，グループ割り込み名を意味します。

[引数]

なし

[戻り値]

なし

3.2.2 クロック発生回路

以下に，コード生成ツールがクロック発生回路用として出力する API 関数の一覧を示します。

表 3.2 クロック発生回路用 API 関数

API 関数名	機能概要
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_cgc_oscillation_stop_interrupt	発振停止検出割り込みの発生に伴う処理を行います。
r_cgc_oscillation_stop_nmi_interrupt	発振停止検出 NMI の発生に伴う処理を行います。
R_CGC_Set_ClockMode	クロック・ソースを設定します。

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。

[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_cgc_oscillation_stop_interrupt

発振停止検出割り込みの発生に伴う処理を行います。

備考 本 API 関数は，クロック発生回路がメイン・クロックの発振停止を検出した場合に発生する発振停止検出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]

```
static  void    r_cgc_oscillation_stop_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_cgc_oscillation_stop_nmi_interrupt

発振停止検出 NMI の発生に伴う処理を行います。

[指定形式]

```
static  void    r_cgc_oscillation_stop_nmi_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_CGC_Set_ClockMode

クロック・ソースを設定します。

[指定形式]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_cgc.h"
MD_STATUS   R_CGC_Set_ClockMode ( clock_mode_t mode );
```

[引数]

I/O	引数	説明
I	clock_mode_t *mode*;	クロック・ソースの種類 MAINCLK : メイン・クロック発振器 SUBCLK : サブクロック発振器 PLLCLK : PLL 回路 HOCO : 高速オンチップ・オシレータ LOCO : 低速オンチップ・オシレータ

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR1	異常終了
MD_ARGERROR	引数 *mode* の指定が不正

3.2.3 電圧検出回路（LVD）

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 3.3 電圧検出回路用 API 関数

API 関数名	機能概要
R_LVDn_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_LVDn_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_lvd_lvdn_interrupt	電圧監視 *n* 割り込みの発生に伴う処理を行います。
R_LVDn_Start	電圧の監視を開始します（割り込みモード，および割り込み & リセット・モード）。
R_LVDn_Stop	電圧の監視を終了します（割り込みモード，および割り込み & リセット・モード）。

R_LVD*n*_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。

[指定形式]

```
void    R_LVDn_Create ( void );
```

備考　　*n* は，回路番号を意味します。

[引数]

なし

[戻り値]

なし

R_LVD*n*_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_LVDn_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_LVDn_Create_UserInit ( void );
```

備考 *n* は，回路番号を意味します。

[引数]

なし

[戻り値]

なし

r_lvd_lvd*n*_interrupt

電圧監視 *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，電圧検出回路が電圧の低下を検出した場合に発生する電圧監視 *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]

```
static  void    r_lvd_lvdn_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

R_LVD*n*_Start

電圧の監視を開始します（割り込みモード，および割り込み & リセット・モード）。

[指定形式]

```
void    R_LVDn_Start ( void );
```

備考　　*n* は，回路番号を意味します。

[引数]

なし

[戻り値]

なし

R_LVD*n*_Stop

電圧の監視を終了します（割り込みモード，および割り込み & リセット・モード）。

[指定形式]

```
void    R_LVDn_Stop ( void );
```

備考　　*n* は，回路番号を意味します。

[引数]

なし

[戻り値]

なし

3.2.4 クロック周波数精度測定回路（CAC）

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 3.4 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_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_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_cac_mendf_interrupt	測定終了割り込みの発生に伴う処理を行います。
r_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。
r_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。
R_CAC_Start	クロック周波数の精度測定を開始します。
R_CAC_Stop	クロック周波数の精度測定を終了します。

R_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。

[指定形式]

```
void    R_CAC_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_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。

備考 本 API 関数は，R_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_CAC_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_cac_mendf_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]

```
static  void    r_cac_mendf_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_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。

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]

```
static  void    r_cac_ferrf_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_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。

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]

```
static  void    r_cac_ovff_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_CAC_Start

クロック周波数の精度測定を開始します。

[指定形式]

```
void    R_CAC_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_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_CAC_Stop ( void );
```

[引数]

なし

[戻り値]

なし

3.2.5 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 3.5 消費電力低減機能用 API 関数

API 関数名	機能概要
R_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_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_LPC_AllModuleClockStop	全モジュールのクロックを停止します。
R_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_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_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_LPC_DeepSoftwareStandby	MCU の低消費電力状態をディープ・ソフトウエア・スタンバイ・モードへと遷移させます。
R_LPC_SoftwareStandby	MCU の低消費電力状態をソフトウエア・スタンバイ・モードへと遷移させます。
R_LPC_ChangeOperatingPowerControl	MCU の動作電力制御状態を変更します。

R_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。

[指定形式]

```
void    R_LPC_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_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_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_LPC_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

R_LPC_AllModuleClockStop

全モジュールのクロックを停止します。

[指定形式]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_LPC_AllModuleClockStop ( 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_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]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_lpc.h"
MD_STATUS   R_LPC_ChangeSleepModeReturnClock ( return_clock_t clock );
```

[引数]

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

[戻り値]

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 *clock* の指定が不正

R_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]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_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_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]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_LPC_DeepSleep ( 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_LPC_DeepSoftwareStandby

ディープ・ソフトウェア・スタンバイ・モードに遷移します。

[指定形式]

```
MD_STATUS R_LPC_DeepSoftwareStandby ( 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_LPC_SoftwareStandby

MCU の低消費電力状態をソフトウエア・スタンバイ・モードへと遷移させます。

[指定形式]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_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_LPC_ChangeOperatingPowerControl

MCU の動作電力制御状態を変更します。

[指定形式]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_lpc.h"
MD_STATUS   R_LPC_ChangeOperatingPowerControl ( operating_mode_t mode );
```

[引数]

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

[戻り値]

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_ERROR2	中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

3.2.6 割り込みコントローラ（ICU）

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 3.6 割り込みコントローラ用 API 関数

API 関数名	機能概要
R_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_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_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_icu_software_interrupt	ソフトウエア割り込みの発生に伴う処理を行います。
r_icu_software2_interrupt	ソフトウエア割り込み 2 の発生に伴う処理を行います。
r_icu_nmi_interrupt	NMI 端子割り込みの発生に伴う処理を行います。
R_ICU_IRQn_Start	外部端子割り込みの検出を許可します。
R_ICU_IRQn_Stop	外部端子割り込みの検出を禁止します。
R_ICU_Software_Start	ソフトウエア割り込みの検出を許可します。
R_ICU_Software2_Start	ソフトウエア割り込み 2 の検出を許可します。
R_ICU_Software_Stop	ソフトウエア割り込みの検出を禁止します。
R_ICU_Software2_Stop	ソフトウエア割り込み 2 の検出を禁止します。
R_ICU_SoftwareInterrupt_Generate	ソフトウエア割り込みを発生させます。
R_ICU_SoftwareInterrupt2_Generate	ソフトウエア割り込み 2 を発生させます。

R_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。

[指定形式]

```
void    R_ICU_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_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_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_ICU_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_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。

備考 本 API 関数は，外部端子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]

```
static  void    r_icu_irqn_interrupt ( void );
```

備考 *n* は，要因番号を意味します。

[引数]

なし

[戻り値]

なし

r_icu_software_interrupt

ソフトウエ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_ICU_SoftwareInterrupt_Generate を呼び出した場合に発生するソフトウエ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]

```
static  void    r_icu_software_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_icu_software2_interrupt

ソフトウエア割り込み 2 の発生に伴う処理を行います。

備考 本 API 関数は，R_ICU_SoftwareInterrupt2_Generate を呼び出した場合に発生するソフトウエ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]

```
static  void    r_icu_software2_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_icu_nmi_interrupt

NMI 端子割り込みの発生に伴う処理を行います。

備考 本 API 関数は，NMI 端子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]

```
static  void    r_icu_nmi_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_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_ICU_IRQn_Start ( void );
```

備考 *n* は，要因番号を意味します。

[引数]

なし

[戻り値]

なし

R_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_ICU_IRQn_Stop ( void );
```

備考　　*n* は，要因番号を意味します。

[引数]

なし

[戻り値]

なし

R_ICU_Software_Start

ソフトウエア割り込みの検出を許可します。

[指定形式]

```
void    R_ICU_Software_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_ICU_Software2_Start

ソフトウエア割り込み 2 の検出を許可します。

[指定形式]

```
void    R_ICU_Software2_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_ICU_Software_Stop

ソフトウエア割り込みの検出を禁止します。

[指定形式]

```
void    R_ICU_Software_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_ICU_Software2_Stop

ソフトウエア割り込み 2 の検出を禁止します。

[指定形式]

```
void    R_ICU_Software2_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_ICU_SoftwareInterrupt_Generate

ソフトウエ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_icu_software_interrupt が呼び出されます。

[指定形式]

```
void    R_ICU_SoftwareInterrupt_Generate ( void );
```

[引数]

なし

[戻り値]

なし

R_ICU_SoftwareInterrupt2_Generate

ソフトウエア割り込み 2 を発生させます。

備考 本 API 関数の呼び出しに伴い，r_icu_software2_interrupt が呼び出されます。

[指定形式]

```
void    R_ICU_SoftwareInterrupt2_Generate ( void );
```

[引数]

なし

[戻り値]

なし

3.2.7 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 3.7 バス用 API 関数

API 関数名	機能概要
R_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_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_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_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_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_BSC_InitializeSDRAM	SDRAM コントローラの初期化を行います。

R_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。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_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_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_BSC_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_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]

```
static  void    r_bsc_buserr_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_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_BSC_Error_Monitoring_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_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_BSC_Error_Monitoring_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_BSC_InitializeSDRAM

SDRAM コントローラの初期化処理を行います。

[指定形式]

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

[引数]

なし

[戻り値]

なし

3.2.8 DMA コントローラ（DMAC）

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 3.8 DMA コントローラ用 API 関数

API 関数名	機能概要
R_DMAC_Create	DMA コントローラの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_DMACn_Start	チャネル ｎ の DMA 起動を許可します。
R_DMACn_Stop	チャネル ｎ の DMA 起動を禁止します。
R_DMACn_Set_SoftwareTrigger	チャネル n のソフトウェア転送要求をセットします。
R_DMACn_Clear_SoftwareTrigger	チャネル n のソフトウェア転送要求をクリアします。
r_dmac_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。
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。
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。
R_DMAC_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_DMAC_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。

[指定形式]

```
void    R_DMAC_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_DMAC*n*_Start

チャネル *n* の DMA 起動を許可します。

[指定形式]

```
void    R_DMACn_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_DMAC*n*_Stop

チャネル *n* の DMAC 起動を禁止します。

[指定形式]

```
void    R_DMACn_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_DMAC*n*_Set_SoftwareTrigger

チャネル *n* のソフトウェア転送要求をセットします。

[指定形式]

```
void    R_DMACn_Set_SoftwareTrigger ( 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_DMAC*n*_Clear_SoftwareTrigger

チャネル *n* のソフトウェア転送要求をクリアします。

[指定形式]

```
void    R_DMACn_Clear_SoftwareTrigger ( 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_dmac_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。

備考　本 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として呼び出されます。

[指定形式]

```
void    r_dmac_dmacni_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_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 本 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_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_dmacn_callback_transfer_end ( 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_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 本 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_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_dmacn_callback_transfer_escape_end ( 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_DMAC_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_DMAC_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_DMAC_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

3.2.9 データ・トランスファ・コントローラ（DTC）

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 3.9 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_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_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_DTCm_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_DTCm_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_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。

[指定形式]

```
void    R_DTC_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_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_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_DTC_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

R_DTC*m*_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，転送情報番号 *m* に対応した DTC 起動許可レジスタ *n*（DTCER*n*）の 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。
m は転送情報番号を，*n* は割り込みベクタ番号を意味します。

[指定形式]

```
void    R_DTCm_Start ( void );
```

備考 *m* は転送情報番号を意味します。

[引数]

なし

[戻り値]

なし

R_DTC*m*_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，転送情報番号 *m* に対応した DTC 起動許可レジスタ *n*（DTCER*n*）の 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。
m は転送情報番号を，*n* は割り込みベクタ番号を意味します。

[指定形式]

```
void    R_DTCm_Stop ( void );
```

備考 *m* は転送情報番号を意味します。

[引数]

なし

[戻り値]

なし

3.2.10 イベント・リンク・コントローラ（ELC）

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 3.10 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_ELC_Create	イベント・リンク・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_ELC_Create_UserInit	イベント・リンク・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_elc_elsrni_interrupt	イベント割り込み ELSRn の発生に伴う処理を行います。
R_ELC_Start	周辺機能間の連携を開始します。
R_ELC_Stop	周辺機能間の連携を終了します。
R_ELC_GenerateSoftwareEvent	ソフトウエア・イベントを発生させます。
R_ELC_Set_PortBuffern	ポート・バッファに値を書き込みます。
R_ELC_Get_PortBuffern	ポート・バッファの値を読み出します。

R_ELC_Create

イベント・リンク・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。

[指定形式]

```
void    R_ELC_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_ELC_Create_UserInit

イベント・リンク・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_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_ELC_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_elc_elsr*n*i_interrupt

イベント割り込み ELSR*n*i の発生に伴う処理を行います。

備考 本 API 関数は，イベント割り込み ELSR*n*i に対応した割り込み処理として呼び出されます。
n はイベントリンク設定レジスタ番号を意味します。

[指定形式]

```
static  void    r_elc_elsrni_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

R_ELC_Start

周辺機能間の連携を開始します。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_ELC_Stop

周辺機能間の連携を終了します。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_ELC_GenerateSoftwareEvent

ソフトウエア・イベントを発生させます。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_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_ELC_Set_PortBuffern ( uint8_t  value );
```

備考　　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_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_ELC_Get_PortBuffern ( uint8_t * const  value );
```

備考　　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

3.2.11 I/O ポート

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

表 3.11 I/O ポート用 API 関数

API 関数名	機能概要
R_PORT_Create	I/O 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_PORT_Create_UserInit	I/O ポートに関するユーザ独自の初期化処理を行います。

R_PORT_Create

I/O 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。

[指定形式]

```
void    R_PORT_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_PORT_Create_UserInit

I/O 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_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_PORT_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

3.2.12 マルチファンクション・タイマ・パルス・ユニット 2（MTU2）

以下に，コード生成ツールがマルチファンクション・タイマ・パルス・ユニット 2 用として出力する API 関数の一覧を示します。

表 3.12 マルチファンクション・タイマ・パルス・ユニット 2 用 API 関数

API 関数名	機能概要
R_MTU2_Create	マルチファンクション・タイマ・パルス・ユニット 2 を制御するうえで必要となる初期化処理を行います。
R_MTU2_Create_UserInit	マルチファンクション・タイマ・パルス・ユニット 2 に関するユーザ独自の初期化処理を行います。
r_mtu2_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。
r_mtu2_cj_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。
r_mtu2_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。
r_mtu2_cj_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。
r_mtu2_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。
R_MTU2_Cn_Start	16 ビット・タイマのカウントを開始します。
R_MTU2_Cn_Stop	16 ビット・タイマのカウントを終了します。

R_MTU2_Create

マルチファンクション・タイマ・パルス・ユニット 2 を制御するうえで必要となる初期化処理を行います。

[指定形式]

```
void    R_MTU2_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_MTU2_Create_UserInit

マルチファンクション・タイマ・パルス・ユニット 2 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_MTU2_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_MTU2_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_mtu2_tgi*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。

備考 本 API 関数は，マルチファンクション・タイマ・パルス・ユニット 2 が入力信号のエッジを検出した場合，または現在カウント値 “ 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インプット・キャプチャ／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]

```
static  void    r_mtu2_tgimn_interrupt ( void );
```

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

[引数]

なし

[戻り値]

なし

r_mtu2_c*j*_tgi*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。

備考 本 API 関数は，マルチファンクション・タイマ・パルス・ユニット 2 が入力信号のエッジを検出した場合，または現在カウント値 “ 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インプット・キャプチャ／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]

```
static  void    r_mtu2_cj_tgimn_interrupt ( void );
```

備考 *j* は関係するチャネル番号を，*m* はタイマ・ジェネラル・レジスタ番号を，*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

r_mtu2_tciv*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。

備考 本 API 関数は，タイマ・カウンタ（TCNT）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]

```
static  void    r_mtu2_tciv*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_mtu2_c*j*_tciv*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。

備考 本 API 関数は，タイマ・カウンタ（TCNT）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]

```
static  void    r_mtu2_cj_tcivn_interrupt ( void );
```

備考 *j* は関係するチャネル番号を，*n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

r_mtu2_tciu*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。

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]

```
static  void    r_mtu2_tciun_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_MTU2_C*n*_Start

16 ビット・タイマのカウントを開始します。

[指定形式]

```
void    R_MTU2_Cn_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_MTU2_C*n*_Stop

16 ビット・タイマのカウントを終了します。

[指定形式]

```
void    R_MTU2_Cn_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

3.2.13 マルチファンクション・タイマ・パルス・ユニット 3（MTU3）

以下に，コード生成ツールがマルチファンクション・タイマ・パルス・ユニット 3 用として出力する API 関数の一覧を示します。

表 3.13 マルチファンクション・タイマ・パルス・ユニット 3 用 API 関数

API 関数名	機能概要
R_MTU3_Create	マルチファンクション・タイマ・パルス・ユニット 3 を制御するうえで必要となる初期化処理を行います。
R_MTU3_Create_UserInit	マルチファンクション・タイマ・パルス・ユニット 3 に関するユーザ独自の初期化処理を行います。
r_mtu3_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。
r_mtu3_cj_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
r_mtu3_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。
r_mtu3_cj_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。
r_mtu3_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。
R_MTU3_Cn_Start	16 ビット・タイマのカウントを開始します。
R_MTU3_Cn_Stop	16 ビット・タイマのカウントを終了します。

R_MTU3_Create

マルチファンクション・タイマ・パルス・ユニットの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。

[指定形式]

```
void    R_MTU3_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_MTU3_Create_UserInit

マルチファンクション・タイマ・パルス・ユニット 3 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_MTU3_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_MTU3_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_mtu3_tgi*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。

備考 本 API 関数は，マルチファンクション・タイマ・パルス・ユニット 3 が入力信号のエッジを検出した場合，または現在カウント値 “ 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インプット・キャプチャ／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]

```
static  void    r_mtu3_tgimn_interrupt ( void );
```

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

[引数]

なし

[戻り値]

なし

r_mtu3_c*j*_tgi*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。

備考 本 API 関数は，マルチファンクション・タイマ・パルス・ユニット 3 が入力信号のエッジを検出した場合，または現在カウント値 “ 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インプット・キャプチャ／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]

```
static  void    r_mtu3_cj_tgimn_interrupt ( void );
```

備考 *j* は関係するチャネル番号，*m* はタイマ・ジェネラル・レジスタ番号を，*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

r_mtu3_tciv*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。

備考 本 API 関数は，タイマ・カウンタ（TCNT）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]

```
static  void    r_mtu3_tciv 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_mtu3_c*j*_tciv*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。

備考 本 API 関数は，タイマ・カウンタ（TCNT）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]

```
static  void    r_mtu3_cj_tcivn_interrupt ( void );
```

備考 *j* は関係するチャネル番号，*n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

r_mtu3_tciu*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。

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]

```
static  void    r_mtu3_tciun_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_MTU3_C*n*_Start

16 ビット・タイマのカウントを開始します。

[指定形式]

```
void    R_MTU3_Cn_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_MTU3_C*n*_Stop

16 ビット・タイマのカウントを終了します。

[指定形式]

```
void    R_MTU3_Cn_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

3.2.14 ポート・アウトプット・イネーブル 2（POE2）

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 2 用として出力する API 関数の一覧を示します。

表 3.14 ポート・アウトプット・イネーブル 2 用 API 関数

API 関数名	機能概要
R_POE2_Create	ポート・アウトプット・イネーブル 2 を制御するうえで必要となる初期化処理を行います。
R_POE2_Create_UserInit	ポート・アウトプット・イネーブル 2 に関するユーザ独自の初期化処理を行います。
r_poe2_oein_interrupt	アウトプット・イネーブル割り込み *n*（OEI*n*）の発生に伴う処理を行います。
R_POE2_Start	MTU 相補 PWM 出力端子をハイインピーダンスとします。
R_POE2_Stop	MTU 相補 PWM 出力端子のハイインピーダンスを解除します。
R_POE2_Set_HiZ_MTUn	MTU*n* 端子をハイインピーダンス状態にします。
R_POE2_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_POE2_Create

ポート・アウトプット・イネーブル 2 を制御するうえで必要となる初期化処理を行います。

[指定形式]

```
void    R_POE2_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_POE2_Create_UserInit

ポート・アウトプット・イネーブル 2 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_POE2_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_POE2_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_poe2_oei*n*_interrupt

アウトプット・イネーブル割り込み *n*（OEI*n*）の発生に伴う処理を行います。

備考 本 API 関数は，端子（POE0#，POE1#，POE2#，POE3#，POE8# のいずれか）がハイインピーダンスとなった場合，または出力短絡フラグ 1 がセットされた場合に発生するアウトプット・イネーブル割り込み *n*（OEI*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]

```
static  void    r_poe2_oein_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_POE2_Start

MTU 相補 PWM 出力端子をハイインピーダンスとします。

[指定形式]

```
void    R_POE2_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_POE2_Stop

MTU 相補 PWM 出力端子のハイインピーダンスを解除します。

[指定形式]

```
void    R_POE2_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_POE2_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_POE2_Set_HiZ_MTUn ( 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_POE2_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_POE2_Clear_HiZ_MTUn ( 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

3.2.15 ポート・アウトプット・イネーブル 3（POE3）

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 3 用として出力する API 関数の一覧を示します。

表 3.15 ポート・アウトプット・イネーブル 3 用 API 関数

API 関数名	機能概要
R_POE3_Create	ポート・アウトプット・イネーブル 3 を制御するうえで必要となる初期化処理を行います。
R_POE3_Create_UserInit	ポート・アウトプット・イネーブル 3 に関するユーザ独自の初期化処理を行います。
r_poe3_oein_interrupt	アウトプット・イネーブル割り込み *n*（OEI*n*）の発生に伴う処理を行います。
R_POE3_Start	MTU 相補 PWM 出力端子をハイインピーダンスとします。
R_POE3_Stop	MTU 相補 PWM 出力端子のハイインピーダンスを解除します。
R_POE3_Set_HiZ_MTUn	MTU*n* 端子をハイインピーダンス状態にします。
R_POE3_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_POE3_Set_HiZ_GPTn	GPT*n* 端子をハイインピーダンス状態にします。
R_POE3_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_POE3_Create

ポート・アウトプット・イネーブル 3 を制御するうえで必要となる初期化処理を行います。

[指定形式]

```
void    R_POE3_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_POE3_Create_UserInit

ポート・アウトプット・イネーブル 3 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_POE3_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_POE3_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_poe3_oei*n*_interrupt

アウトプット・イネーブル割り込み *n*（OEI*n*）の発生に伴う処理を行います。

備考 本 API 関数は，アウトプットイネーブル割り込み OEI*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]

```
static  void    r_poe3_oein_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

R_POE3_Start

端子をソフトウェアトリガによりハイインピーダンス状態にします。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_POE3_Stop

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_POE3_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_POE3_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_POE3_Set_HiZ_MTUn ( 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_POE3_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_POE3_Clear_HiZ_MTUn ( 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_POE3_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_POE3_Set_HiZ_GPTn ( 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_POE3_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_POE3_Clear_HiZ_GPTn ( 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

3.2.16 汎用 PWM タイマ（GPT）

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 3.16 汎用 PWM タイマ用 API 関数

API 関数名	機能概要
R_GPT_Create	汎用 PWM タイマの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_GPTn_Start	チャネル n のカウントを開始します。
R_GPTn_Stop	チャネル n のカウントを終了します。
R_GPTn_HardwareStart	チャネル n に関する割り込みを許可します。
R_GPTn_HardwareStop	チャネル n に関する割り込みを禁止します。
R_GPT_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_gpt_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_gpt_gtcivn_interrupt	オーバーフロー割り込みの発生に伴う処理を行います。
r_gpt_gtciun_interrupt	アンダーフロー割り込みの発生に伴う処理を行います。
r_gpt_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_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_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_Create

汎用 PWM タイマの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。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_GPT*n*_Start

チャネル *n* のカウントを開始します。

[指定形式]

```
void    R_GPTn_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_GPT*n*_Stop

チャネル *n* のカウントを終了します。

[指定形式]

```
void    R_GPTn_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_GPT*n*_HardwareStart

チャネル *n* に関する割り込みを有効化します。

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_GPTn_HardwareStart ( 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*n*_HardwareStop

チャネル *n* に関する割り込みを無効化します。

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無効化するために使用します。

[指定形式]

```
void    R_GPTn_HardwareStop ( 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_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_GPT_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_GPT_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_gpt_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]

```
static void    r_gpt_gtcimn_interrupt ( void );
```

備考　　*m* はジェネラルレジスタ番号を，*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_gtciv*n*_interrupt

オーバーフロー割り込みの発生に伴う処理を行います。

[指定形式]

```
static void    r_gpt_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_gpt_gtciu*n*_interrupt

アンダーフロー割り込みの発生に伴う処理を行います。

[指定形式]

```
static void    r_gpt_gtciun_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_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]

```
static void    r_gpt_gdten_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_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]

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

[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]

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

[引数]

なし

[戻り値]

なし

3.2.17 16 ビットタイマ・パルス・ユニット（TPU）

以下に，コード生成ツールが 16 ビットタイマ・パルス・ユニット用として出力する API 関数の一覧を示します。

表 3.17 16 ビットタイマ・パルス・ユニット用 API 関数

API 関数名	機能概要
R_TPU_Create	16 ビットタイマ・パルス・ユニットの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_TPUn_Start	チャネル n のカウントを開始します。
R_TPUn_Stop	チャネル n のカウントを終了します。
R_TPU_Create_UserInit	16 ビットタイマ・パルス・ユニットに関するユーザ独自の初期化処理を行います。
r_tpu_tginm_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_tpu_tcinv_interrupt	オーバーフロー割り込みの発生に伴う処理を行います。
r_tpu_tcinu_interrupt	アンダーフロー割り込みの発生に伴う処理を行います。

R_TPU_Create

16 ビットタイマ・パルス・ユニットの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。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_TPU*n*_Start

チャネル *n* のカウントを開始します。

[指定形式]

```
void    R_TPUn_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_TPU*n*_Stop

チャネル *n* のカウントを終了します。

[指定形式]

```
void    R_TPUn_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_TPU_Create_UserInit

16 ビットタイマ・パルス・ユニッ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_TPU_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_TPU_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_tpu_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]

```
static  void    r_tpu_tgi nm _interrupt ( void );
```

備考　　*m* はジェネラルレジスタ番号を，*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

r_tpu_tci*n*v_interrupt

オーバーフロー割り込みの発生に伴う処理を行います。

[指定形式]

```
static  void    r_tpu_tcinv_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_tpu_tci*n*u_interrupt

アンダーフロー割り込みの発生に伴う処理を行います。

[指定形式]

```
static  void    r_tpu_tcinu_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

3.2.18 8 ビットタイマ・パルス・ユニット（TMR）

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 3.18 8 ビットタイマ用 API 関数

API 関数名	機能概要
R_TMR_Create	8 ビットタイマの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_TMRn_Start	チャネル n のカウントを開始します。
R_TMRn_Stop	チャネル n のカウントを終了します。
R_TMR_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_tmr_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_tmr_ovin_interrupt	オーバーフロー割り込みの発生に伴う処理を行います。

R_TMR_Create

8 ビットタイマの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。

[指定形式]

```
void    R_TMR_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_TMR*n*_Start

チャネル *n* のカウントを開始します。

[指定形式]

```
void    R_TMRn_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_TMR*n*_Stop

チャネル *n* のカウントを終了します。

[指定形式]

```
void    R_TMRn_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_TMR_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_TMR_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_TMR_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_tmr_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]

```
static  void    r_tmr_cmimn_interrupt ( void );
```

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

[引数]

なし

[戻り値]

なし

r_tmr_ovi*n*_interrupt

オーバーフロー割り込みの発生に伴う処理を行います。

[指定形式]

```
static  void    r_tmr_ovin_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

3.2.19 プログラマブル・パルスジェネレータ（PPG）

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 3.19 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_PPG_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_PPG_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_PPG_Create

プログラマブル・パルスジェネレータの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。

[指定形式]

```
void    R_PPG_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_PPG_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_PPG_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_PPG_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

3.2.20 コンペア・マッチ・タイマ（CMT）

以下に，コード生成ツールがコンペア・マッチ・タイマ用として出力する API 関数の一覧を示します。

表 3.20 コンペア・マッチ・タイマ用 API 関数

API 関数名	機能概要
R_CMTn_Create	コンペア・マッチ・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_CMTn_Create_UserInit	コンペア・マッチ・タイマに関するユーザ独自の初期化処理を行います。
r_cmt_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_CMTn_Start	16 ビット・タイマのカウントを開始します。
R_CMTn_Stop	16 ビット・タイマのカウントを終了します。

R_CMT*n*_Create

コンペア・マッチ・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。

[指定形式]

```
void    R_CMTn_Create ( 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_CMT*n*_Create_UserInit

コンペア・マッチ・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_CMTn_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_CMTn_Create_UserInit ( 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_cmt_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]

```
static  void    r_cmt_cmin_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_CMT*n*_Start

16 ビット・タイマのカウントを開始します。

[指定形式]

```
void    R_CMTn_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_CMT*n*_Stop

16 ビット・タイマのカウントを終了します。

[指定形式]

```
void    R_CMTn_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

3.2.21 コンペア・マッチ・タイマ W（CMTW）

以下に，コード生成ツールがコンペア・マッチ・タイマ W 用として出力する API 関数の一覧を示します。

表 3.21 コンペア・マッチ・タイマ W 用 API 関数

API 関数名	機能概要
R_CMTWn_Create	コンペア・マッチ・タイマ W の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_CMTWn_Start	チャネル n のカウントを開始します。
R_CMTWn_Stop	チャネル n のカウントを終了します。
R_CMTWn_Create_UserInit	コンペア・マッチ・タイマ W に関するユーザ独自の初期化処理を行います。
r_cmtw_cmwin_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_cmtw_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_cmtw_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_CMTW*n*_Create

コンペア・マッチ・タイマ W の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。

[指定形式]

```
void    R_CMTWn_Create ( 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_CMTW*n*_Start

チャネル *n* のカウントを開始します。

[指定形式]

```
void    R_CMTWn_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_CMTW*n*_Stop

チャネル *n* のカウントを終了します。

[指定形式]

```
void    R_CMTWn_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_CMTW*n*_Create_UserInit

コンペア・マッチ・タイマ W 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_CMTWn_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_CMTWn_Create_UserInit ( 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_cmtw_cmwi*n*_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]

```
static  void    r_cmtw_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_cmtw_ic*min*_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]

```
static void    r_cmtw_icmin_interrupt ( void );
```

備考　　*m* はインプット・キャプチャ・レジスタ番号を，*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

r_cmtw_oc*min*_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]

```
static void    r_cmtw_oc min _interrupt ( void );
```

備考　　*m* はアウトプット・コンペア・レジスタ番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

3.2.22 リアルタイム・クロック（RTC）

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 3.22 リアルタイム・クロック用 API 関数

API 関数名	機能概要
R_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_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_rtc_alm_interrupt	アラーム割り込み（ALM）の発生に伴う処理を行います。
r_rtc_prd_interrupt	周期割り込み（PRD）の発生に伴う処理を行います。
r_rtc_cup_interrupt	桁上げ割り込み（CUP）の発生に伴う処理を行います。
R_RTC_Set_CalendarAlarm	アラーム割り込み（ALM）の発生条件を設定すると伴に，アラーム割り込み（ALM）の検出を許可します（カレンダ・カウント・モード）。
R_RTC_Set_BinaryAlarm	アラーム割り込み（ALM）の発生条件を設定すると伴に，アラーム割り込み（ALM）の検出を許可します（バイナリ・カウント・モード）。
R_RTC_Set_ConstPeriodInterruptOn	周期割り込み（PRD）の発生周期を設定すると伴に，周期割り込み（PRD）の検出を許可します。
R_RTC_Set_ConstPeriodInterruptOff	周期割り込み（PRD）の検出を禁止します。
R_RTC_Set_CarryInterruptOn	桁上げ割り込み（CUP）の検出を許可します。
R_RTC_Set_CarryInterruptOff	桁上げ割り込み（CUP）の検出を禁止します。
R_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_RTC_Set_RTCOUTOff	RTCOUT 出力を終了します。
R_RTC_Start	カウントを開始します。
R_RTC_Stop	カウントを終了します。
R_RTC_Restart	カウンタを初期化したのち，カウントを開始します。
R_RTC_Set_CalendarCounterValue	カレンダ値を設定します。
R_RTC_Get_CalendarCounterValue	カレンダ値を獲得します。
R_RTC_Set_BinaryCounterValue	バイナリ・カウント値を設定します。
R_RTC_Get_BinaryCounterValue	バイナリ・カウント値を獲得します。
R_RTC_Get_CalendarTimeCaptureValuen	キャプチャしたカレンダ値を獲得します。
R_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_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。

[指定形式]

```
void    R_RTC_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_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_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_RTC_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_rtc_alm_interrupt

アラーム割り込み（ALM）の発生に伴う処理を行います。

備考 本 API 関数は，R_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（ALM）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]

```
static  void    r_rtc_alm_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_rtc_prd_interrupt

周期割り込み（PRD）の発生に伴う処理を行います。

備考 本 API 関数は，R_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（PRD）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]

```
static  void    r_rtc_prd_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_rtc_cup_interrupt

桁上げ割り込み（CUP）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（CUP）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]

```
static  void    r_rtc_cup_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_RTC_Set_CalendarAlarm

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

[指定形式]

```
#include    "r_cg_rtc.h"
void    R_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_RTC_Set_BinaryAlarm

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

[指定形式]

```
#include    "r_cg_macrodriver.h"
void    R_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_RTC_Set_ConstPeriodInterruptOn

周期割り込み（PRD）の発生周期を設定すると伴に，周期割り込み（PRD）の検出を許可します。

[指定形式]

```
#include    "r_cg_rtc.h"
void    R_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（PRD）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_RTC_Set_ConstPeriodInterruptOff

周期割り込み（PRD）の検出を禁止します。

[指定形式]

```
void    R_RTC_Set_ConstPeriodInterruptOff ( void );
```

[引数]

なし

[戻り値]

なし

R_RTC_Set_CarryInterruptOn

桁上げ割り込み（CUP）の検出を許可します。

[指定形式]

```
void    R_RTC_Set_CarryInterruptOn ( void );
```

[引数]

なし

[戻り値]

なし

R_RTC_Set_CarryInterruptOff

桁上げ割り込み（CUP）の検出を禁止します。

[指定形式]

```
void    R_RTC_Set_CarryInterruptOff ( void );
```

[引数]

なし

[戻り値]

なし

R_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]

```
#include    "r_cg_rtc.h"
void    R_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_RTC_Set_RTCOUTOff

RTCOUT 出力を終了します。

[指定形式]

```
void    R_RTC_Set_RTCOUTOff ( void );
```

[引数]

なし

[戻り値]

なし

R_RTC_Start

カウントを開始します。

[指定形式]

```
void    R_RTC_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_RTC_Stop

カウントを終了します。

[指定形式]

```
void    R_RTC_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_RTC_Restart

カウンタを初期化したのち，カウントを開始します。

備考 1. 本 API 関数では，リアルタイム・クロックがカレンダ・カウント・モードで動作している際は，カウンタの初期化を引数 *counter_write_val* で指定された値で行います。

備考 2. 本 API 関数では，リアルタイム・クロックがバイナリ・カウント・モードで動作している際は，引数 *counter_write_val* で指定された値を無視し，カウンタをゼロ・クリアします。

[指定形式]

```
#include    "r_cg_rtc.h"
void    R_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_RTC_Set_CalendarCounterValue

カレンダ値を設定します。

[指定形式]

```
#include    "r_cg_rtc.h"
void    R_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_RTC_Get_CalendarCounterValue

カレンダ値を獲得します。

[指定形式]

```
#include    "r_cg_rtc.h"
void    R_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_RTC_Set_BinaryCounterValue

バイナリ・カウント値を設定します。

[指定形式]

```
#include    "r_cg_macrodriver.h"
void    R_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_RTC_Get_BinaryCounterValue

バイナリ・カウント値を獲得します。

[指定形式]

```
#include    "r_cg_macrodriver.h"
void    R_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_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_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*;	カウント値へのポインタ

備考　　以下に，リアルタイムクロックのカウント値 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_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_RTC_Get_BinaryTimeCaptureValuen ( 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*;	カウント値へのポインタ

[戻り値]

なし

3.2.23 ウォッチドッグ・タイマ（WDT）

以下に，コード生成ツールがウォッチドッグ・タイマ用として出力する API 関数の一覧を示します。

表 3.23 ウォッチドッグ・タイマ用 API 関数

API 関数名	機能概要
R_WDT_Create	ウォッチドッグタイマの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_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_WDT_Create_UserInit	ウォッチドッグタイマに関するユーザ独自の初期化処理を行います。
r_wdt_wuni_interrupt	マスカブル割り込み / 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_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_WDT_Create

ウォッチドッグ・タイマの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。

[指定形式]

```
void    R_WDT_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_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_WDT_Restart ( void );
```

[引数]

なし

[戻り値]

なし

R_WDT_Create_UserInit

ウォッチドッグ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_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]

```
static  void    R_WDT_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_wdt_wuni_interrupt

マスカブル割り込み / 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 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]

```
static  void    r_wdt_wuni_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_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 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]

```
static  void    r_wdt_nmi_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

3.2.24 独立ウォッチドッグ・タイマ（IWDT）

以下に，コード生成ツールが独立ウォッチドッグ・タイマ用として出力する API 関数の一覧を示します。

表 3.24 独立ウォッチドッグ・タイマ用 API 関数

API 関数名	機能概要
R_IWDT_Create	独立ウォッチドッグ・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_IWDT_Create_UserInit	独立ウォッチドッグ・タイマに関するユーザ独自の初期化処理を行います。
r_iwdt_nmi_interrupt	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_iwdt_iwuni_interrupt	マスカブル割り込み / 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_IWDT_Restart	独立ウォッチドッグ・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_IWDT_Create

独立ウォッチドッグ・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。

[指定形式]

```
void    R_IWDT_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_IWDT_Create_UserInit

独立ウォッチドッグ・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_IWDT_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_IWDT_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_iwdt_nmi_interrupt

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した場合，またはリフレッシュ許可期間以外でリフレッシュを行った場合に発生するノンマスカブル割り込み（WUNI）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］

```
static  void    r_iwdt_nmi_interrupt ( void );
```

［引数］

なし

［戻り値］

なし

r_iwdt_iwuni_interrupt

マスカブル割り込み / 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 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]

```
static  void    r_iwdt_iwuni_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_IWDT_Restart

独立ウォッチドッグ・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_IWDT_Restart ( void );
```

[引数]

なし

[戻り値]

なし

3.2.25 シリアル・コミュニケーション・インタフェース（SCI）

以下に，コード生成ツールがシリアル・コミュニケーション・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 3.25 シリアル・コミュニケーション・インタフェース用 API 関数

API 関数名	機能概要
R_SCIn_Create	シリアル・コミュニケーション・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_SCIn_Create_UserInit	シリアル・コミュニケーション・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_scin_transmitend_interrupt	送信終了割り込みの発生に伴う処理を行います。
r_scin_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_scin_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_scin_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_SCIn_Start	SCI 通信を開始します。
R_SCIn_Stop	SCI 通信を終了します。
R_SCIn_Serial_Send	SCI 送信を開始します（調歩同期式モード）。
R_SCIn_Serial_Receive	SCI 受信を開始します（調歩同期式モード）。
R_SCIn_Serial_Multiprocessor_Send	SCI 送信を開始します（マルチプロセッサ通信機能）。
R_SCIn_Serial_Multiprocessor_Receive	SCI 受信を開始します（マルチプロセッサ通信機能）。
R_SCIn_Serial_Send_Receive	SCI 送受信を開始します（クロック同期式モード）。
R_SCIn_SmartCard_Send	SCI 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_SCIn_SmartCard_Receive	SCI 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_SCIn_IIC_Master_Send	SCI マスタ送信を開始します（簡易 I^2C モード）。
R_SCIn_IIC_Master_Receive	SCI マスタ受信を開始します（簡易 I^2C モード）。
R_SCIn_SPI_Master_Send	SCI マスタ送信を開始します（簡易 SPI モード）。
R_SCIn_SPI_Master_Send_Receive	SCI マスタ送受信を開始します（簡易 SPI モード）。
R_SCIn_SPI_Slave_Send	SCI スレーブ送信を開始します（簡易 SPI モード）。
R_SCIn_SPI_Slave_Send_Receive	SCI スレーブ送受信を開始します（簡易 SPI モード）。
R_SCIn_IIC_StartCondition	スタート・コンディションを発行します。（簡易 I^2C モード）
R_SCIn_IIC_StopCondition	ストップ・コンディションを発行します。（簡易 I^2C モード）
r_scin_callback_transmitend	送信終了割り込みの発生に伴う処理を行います。
r_scin_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_scin_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_SCI*n*_Create

シリアル・コミュニケーション・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。

[指定形式]

```
void    R_SCIn_Create ( 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_SCI*n*_Create_UserInit

シリアル・コミュニケーション・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_SCIn_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_SCIn_Create_UserInit ( 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_sci*n*_transmitend_interrupt

送信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，送信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]

```
static  void    r_sci n _transmitend_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_sci*n*_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。

備考 本 API 関数は，送信データエンプティ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]

```
static  void    r_sci n_transmit_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_sci*n*_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。

備考　本 API 関数は，受信データフル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]

```
static  void    r_sci n_receive_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_sci*n*_receiveerror_interrupt

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，受信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]

```
static  void    r_sci n_receiveerror_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_SCI*n*_Start

SCI 通信を開始します。

[指定形式]

```
void    R_SCIn_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_SCI*n*_Stop

SCI 通信を終了します。

[指定形式]

```
void    R_SCIn_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_SCI*n*_Serial_Send

SCI 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 SCI 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. SCI 送信を行う際には，本 API 関数の呼び出し以前に R_SCIn_Start を呼び出す必要があります。

［指定形式］

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_Serial_Send ( uint8_t * const tx_buf, uint16_t tx_num );
```

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	`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_SCI*n*_Serial_Receive

SCI 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 SCI 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. SCI 受信を行う際には，本 API 関数の呼び出し以前に R_SCIn_Start を呼び出す必要があります。

[指定形式]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_Serial_Receive ( uint8_t * const rx_buf, uint16_t rx_num );
```

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	`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_SCI*n*_Serial_Multiprocessor_Send

SCI 送信を開始します（マルチプロセッサ通信機能）。

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 SCI 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. SCI 送信を行う際には，本 API 関数の呼び出し以前に R_SCIn_Start を呼び出す必要があります。

[指定形式]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_Serial_Multiprocessor_Send ( uint8_t * id_buf, uint16_t id_num,
uint8_t * const tx_buf, uint16_t tx_num );
```

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	uint8_t * *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_SCI*n*_Serial_Multiprocessor_Receive

SCI 受信を開始します（マルチプロセッサ通信機能）。

備考 1. 本 API 関数では，1 バイト単位の SCI 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. SCI 受信を行う際には，本 API 関数の呼び出し以前に R_SCIn_Start を呼び出す必要があります。

[指定形式]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_Serial_Multiprocessor_Receive ( uint8_t * const rx_buf, uint16_t
rx_num );
```

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	`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_SCI*n*_Serial_Send_Receive

SCI 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，SCI 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 SCI 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，SCI 受信処理として，1 バイト単位の SCI 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. SCI 送受信を行う際には，本 API 関数の呼び出し以前に R_SCIn_Start を呼び出す必要があります。

[指定形式]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_Serial_Send_Receive ( uint8_t * const tx_buf, uint16_t tx_num,
uint8_t * const rx_buf, uint16_t rx_num );
```

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	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_SCI*n*_SmartCard_Send

SCI 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 SCI 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. SCI 送信を行う際には，本 API 関数の呼び出し以前に R_SCIn_Start を呼び出す必要があります。

[指定形式]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_SmartCard_Send ( uint8_t * const tx_buf, uint16_t tx_num );
```

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	`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_SCI*n*_SmartCard_Receive

SCI 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 SCI 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. SCI 受信を行う際には，本 API 関数の呼び出し以前に R_SCIn_Start を呼び出す必要があります。

[指定形式]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_SmartCard_Receive ( uint8_t * const rx_buf, uint16_t rx_num );
```

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	`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_SCI*n*_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 SCI 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 SCI 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，SCI マスタ送信の開始処理として，内部的に R_SCIn_IIC_StartCondition の呼び出しを行っています。

備考 3. SCI マスタ送信を行う際には，本 API 関数の呼び出し以前に R_SCIn_Start を呼び出す必要があります。

[指定形式]

```
#include    "r_cg_macrodriver.h"
void    R_SCIn_IIC_Master_Send ( uint8_t adr, uint8_t * const tx_buf, uint16_t tx_num
);
```

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	uint8_t *adr*;	スレーブ・アドレス
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_SCI*n*_IIC_Master_Receive

SCI マスタ受信を開始します（簡易 I^2C 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 SCI マスタ送信したのち，1 バイト単位の SCI 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，SCI マスタ受信の開始処理として，内部的に R_SCIn_IIC_StartCondition の呼び出しを行っています。

備考 3. SCI マスタ受信を行う際には，本 API 関数の呼び出し以前に R_SCIn_Start を呼び出す必要があります。

[指定形式]

```
#include    "r_cg_macrodriver.h"
void    R_SCIn_IIC_Master_Receive ( uint8_t adr, uint8_t * const rx_buf, uint16_t
rx_num );
```

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	uint8_t *adr*;	スレーブ・アドレス
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_SCI*n*_SPI_Master_Send

SCI 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 SCI 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. SCI マスタ送信を行う際には，本 API 関数の呼び出し以前に R_SCIn_Start を呼び出す必要があります。

[指定形式]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_SPI_Master_Send ( uint8_t * const tx_buf, uint16_t tx_num );
```

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	`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_SCI*n*_SPI_Master_Send_Receive

SCI 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，SCI 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 SCI 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，SCI マスタ受信処理として，1 バイト単位の SCI 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. SCI マスタ送受信を行う際には，本 API 関数の呼び出し以前に R_SCIn_Start を呼び出す必要があります。

[指定形式]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_SPI_Master_Send_Receive ( uint8_t * const tx_buf, uint16_t tx_num,
uint8_t * const rx_buf, uint16_t rx_num );
```

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	`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_SCI*n*_SPI_Slave_Send

SCI 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 SCI 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. SCI 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_SCIn_Start を呼び出す必要があります。

[指定形式]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_SPI_Slave_Send ( uint8_t * const tx_buf, uint16_t tx_num );
```

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	`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_SCI*n*_SPI_Slave_Send_Receive

SCI 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，SCI 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 SCI 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，SCI スレーブ受信処理として，1 バイト単位の SCI 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. SCI スレーブ送受信を行う際には，本 API 関数の呼び出し以前に R_SCIn_Start を呼び出す必要があります。

[指定形式]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_SCIn_SPI_Slave_Send_Receive ( uint8_t * const tx_buf, uint16_t tx_num,
uint8_t * const rx_buf, uint16_t rx_num );
```

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	`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_SCI*n*_IIC_StartCondition

スタート・コンディションを発行します。

備考 本 API 関数は，R_SCIn_IIC_Master_Send，および R_SCIn_IIC_Master_Receive の内部関数として呼び出されます。

[指定形式]

```
void    R_SCIn_IIC_StartCondition ( 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_SCI*n*_IIC_StopCondition

ストップ・コンディションを発行します。

[指定形式]

```
void    R_SCIn_IIC_StopCondition ( 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_sci*n*_callback_transmitend

送信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，r_scin_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]

```
static  void    r_sci n _callback_transmitend ( 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_sci*n*_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。

備考 本 API 関数は，r_scin_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]

```
static  void    r_sci n _callback_receiveend ( 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_sci*n*_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_scin_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]

```
static  void    r_sci n _callback_receiveerror ( 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

3.2.26 FIFO 内蔵シリアル・コミュニケーション・インターフェース（SCIFA）

以下に，コード生成ツールが FIFO 内蔵シリアル・コミュニケーション・インターフェース用として出力する API 関数の一覧を示します。

表 3.26 FIFO 内蔵シリアル・コミュニケーション・インターフェース用 API 関数

API 関数名	機能概要
R_SCIFAn_Create	SCIFA の初期化処理を行います。
R_SCIFAn_Start	SCIFA 通信を待機状態にします。
R_SCIFAn_Stop	SCIFA 通信を終了します。
R_SCIFAn_Serial_Send	調歩同期式モードで、送信を開始します。
R_SCIFAn_Serial_Receive	調歩同期式モードで、受信を開始します。
R_SCIFAn_Serial_Send_Receive	クロック同期式モードで、送受信を開始します。
R_SCIFAn_Create_UserInit	SCIFA に関するユーザ独自の初期化処理を行います。
r_scifan_teif_interrupt	トランスミットエンド割り込みの発生に伴う処理を行います。
r_scifan_txif_interrupt	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_scifan_rxif_interrupt	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_scifan_erif_interrupt	フレーミングエラー / パリティエラー割り込みの発生に伴う処理を行います。
r_scifan_brif_interrupt	ブレーク / オーバーラン割り込みの発生に伴う処理を行います。
r_scifan_drif_interrupt	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_scifan_callback_transmitend	トランスミットエンド割り込みの発生に伴う処理を行います。
r_scifan_callback_receiveend	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_scifan_callback_error	エラー割り込みの発生に伴う処理を行います。

R_SCIFA*n*_Create

SCIFA の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。

[指定形式]

```
void    R_SCIFAn_Create ( 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_SCIFA*n*_Start

SCIFA 通信を待機状態にします。

[指定形式]

```
void    R_SCIFAn_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_SCIFA*n*_Stop

SCIFA 通信を終了します。

[指定形式]

```
void    R_SCIFAn_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_SCIFA*n*_Serial_Send

調歩同期式モードで、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. 本 API 関数の呼び出し以前に R_SCIFAn_Start を呼び出す必要があります。

[指定形式]

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

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	`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	引数の指定が不正

R_SCIFA*n*_Serial_Receive

調歩同期式モードで、受信を開始します。

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. 本 API 関数の呼び出し以前に，R_SCIFAn_Start を呼び出す必要があります。

[指定形式]

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

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	`uint8_t * const rx_buf;`	受信するデータを格納した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	引数の指定が不正

R_SCIFA*n*_Serial_Send_Receive

クロック同期式モードで、送受信を開始します。

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. 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. 本 API 関数の呼び出し以前に，R_SCIFAn_Start を呼び出す必要があります。

[指定形式]

```
MD_STATUS   R_SCIFAn_Serial_Send_Receive ( uint8_t * const tx_buf, uint16_t tx_num,
uint8_t * const rx_buf, uint16_t rx_num );
```

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	`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;`	受信するデータを格納した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	引数の指定が不正

R_SCIFA*n*_Create_UserInit

SCIFA 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_SCIFAn_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_SCIFAn_Create_UserInit ( 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_scifa*n*_teif_interrupt

トランスミットエンド割り込みの発生に伴う処理を行います。

備考 本 API 関数は，トランスミットエンド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]

```
static  void    r_scifan_teif_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_scifa*n*_txif_interrupt

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　本 API 関数は，送信 FIFO データエンプティ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]

```
static  void    r_scifan_txif_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_scifa*n*_rxif_interrupt

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 本 API 関数は，受信 FIFO データフル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]

```
static void    r_scifan_rxif_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_scifa*n*_erif_interrupt

フレーミングエラー / パリティ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 / パリティ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]

```
static void    r_scifan_erif_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_scifa*n*_brif_interrupt

ブレーク / オーバーラン割り込みの発生に伴う処理を行います。

備考 API 関数は，ブレーク / オーバーラン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]

```
static void    r_scifan_brif_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_scifa*n*_drif_interrupt

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。

備考　本 API 関数は，受信データレディ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]

```
static void    r_scifan_drif_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_scifa*n*_callback_transmitend

トランスミットエンド割り込みの発生に伴う処理を行います。

備考 本 API 関数は，トランスミットエンド割り込みに対応した割り込み処理 r_scifan_teif_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]

```
static void    r_scifan_callback_trasnmitend ( 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_scifa*n*_callback_receiveend

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 本 API 関数は，受信 FIFO データフル割り込みに対応した割り込み処理 r_scifan_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]

```
static void    r_scifan_callback_receiveend ( 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_scifa*n*_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，エラー割り込みに対応した割り込み処理 r_scifan_erif_interrupt および r_scifan_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]

```
static void    r_scifan_callback_error ( 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

3.2.27 I²C バス・インタフェース（RIIC）

以下に，コード生成ツールが I²C バス・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 3.27 I²C バス・インタフェース用 API 関数

API 関数名	機能概要
R_RIICn_Create	I²C バス・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_RIICn_Create_UserInit	I²C バス・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_riicn_error_interrupt	通信エラー／イベント発生割り込み（EEI）の発生に伴う処理を行います。
r_riicn_receive_interrupt	受信データ・フル割り込み（RXI）の発生に伴う処理を行います。
r_riicn_transmit_interrupt	送信データ・エンプティ割り込み（TXI）の発生に伴う処理を行います。
r_riicn_transmitend_interrupt	送信終了割り込み（TEI）の発生に伴う処理を行います。
R_RIICn_Start	RIIC 通信を開始します。
R_RIICn_Stop	RIIC 通信を終了します。
R_RIICn_Master_Send	RIIC マスタ送信を開始します。
R_RIICn_Master_Receive	RIIC マスタ受信を開始します。
R_RIICn_Slave_Send	RIIC スレーブ送信を開始します。
R_RIICn_Slave_Receive	RIIC スレーブ受信を開始します。
R_RIICn_StartCondition	スタート・コンディションを発行し，通信エラー／イベント発生割り込み（EEI）を発生させます
R_RIICn_StopCondition	ストップ・コンディションを発行し，通信エラー／イベント発生割り込み（EEI）を発生させます。
r_riicn_callback_receiveerror	通信エラー／イベント発生割り込み（EEI）に対応した割り込み処理のうち，アビトレーションロストの検出，NACK の検出，タイムアウトの検出に特化した処理を行います。
r_riicn_callback_transmitend	通信エラー／イベント発生割り込み（EEI）に対応した割り込み処理のうち，R_RIICn_Master_Send の呼び出しに伴うスタート・コンディションの検出に特化した処理を行います。
r_riicn_callback_receiveend	通信エラー／イベント発生割り込み（EEI）に対応した割り込み処理のうち，R_RIICn_Master_Receive の呼び出しに伴うスタート・コンディションの検出に特化した処理を行います。

R_RIIC*n*_Create

I²C バス・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。

[指定形式]

```
void    R_RIICn_Create ( 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_RIIC*n*_Create_UserInit

I^2C バス・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_RIICn_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_RIICn_Create_UserInit ( 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_riic*n*_error_interrupt

通信エラー／イベント発生割り込み（EEI）の発生に伴う処理を行います。

備考 本 API 関数は，I²C バス・インタフェースが通信エラー／イベント発生（アビトレーションロスト，NACK，タイムアウト，スタート・コンディション，ストップ・コンディション）を検出した場合に発生する通信エラー／イベント発生割り込み（EEI）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]

```
static  void    r_riicn_error_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_riic*n*_receive_interrupt

受信データ・フル割り込み（RXI）の発生に伴う処理を行います。

備考　本 API 関数は，受信データ・フル割り込み（RXI）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]

```
static  void    r_riicn_receive_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_riic*n*_transmit_interrupt

送信データ・エンプティ割り込み（TXI）の発生に伴う処理を行います。

備考 本 API 関数は，送信データ・エンプティ割り込み（TXI）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]

```
static  void    r_riicn_transmit_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_riic*n*_transmitend_interrupt

送信終了割り込み（TEI）の発生に伴う処理を行います。

備考 本 API 関数は，送信終了割り込み（TEI）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]

```
static  void    r_riicn_transmitend_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_RIIC*n*_Start

RIIC 通信を開始します。

[指定形式]

```
void    R_RIICn_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_RIIC*n*_Stop

RIIC 通信を終了します。

[指定形式]

```
void    R_RIICn_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_RIIC*n*_Master_Send

RIIC 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 RIIC 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 RIIC 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，RIIC マスタ送信の開始処理として，内部的に R_RIICn_StartCondition の呼び出しを行っています。

備考 3. RIIC マスタ送信を行う際には，本 API 関数の呼び出し以前に R_RIICn_Start を呼び出す必要があります。

[指定形式]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_RIICn_Master_Send ( uint8_t adr, uint8_t * const tx_buf, uint16_t tx_num
);
```

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	`uint8_t adr;`	スレーブ・アドレス
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_RIIC*n*_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）をスレーブ・デバイスに RIIC マスタ送信したのち，1 バイト単位の RIIC 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，RIIC マスタ受信の開始処理として，内部的に R_RIICn_StartCondition の呼び出しを行っています。

備考 3. RIIC マスタ受信を行う際には，本 API 関数の呼び出し以前に R_RIICn_Start を呼び出す必要があります。

[指定形式]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_RIICn_Master_Receive ( uint8_t adr, uint8_t * const rx_buf, uint16_t
rx_num );
```

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	uint8_t *adr*;	スレーブ・アドレス
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_ERROR1	バス・ビジー
MD_ERROR2	引数 *adr* の指定が不正

R_RIIC*n*_Slave_Send

RIIC 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 RIIC 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. RIIC 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_RIICn_Start を呼び出す必要があります。

[指定形式]

```
#include    "r_cg_macrodriver.h"
void    R_RIICn_Slave_Send ( uint8_t * const tx_buf, uint16_t tx_num );
```

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	`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_RIIC*n*_Slave_Receive

RIIC 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 RIIC 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. RIIC 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_RIICn_Start を呼び出す必要があります。

[指定形式]

```
#include    "r_cg_macrodriver.h"
void    R_RIICn_Slave_Receive ( uint8_t * const rx_buf, uint16_t rx_num );
```

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	`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_RIIC*n*_StartCondition

スタート・コンディションを発行し，通信エラー／イベント発生割り込み（EEI）を発生させます。

備考 1. 本 API 関数は，R_RIICn_Master_Send，および R_RIICn_Master_Receive の内部関数として呼び出されます。

備考 2. 本 API 関数の呼び出しに伴い，r_riicn_error_interrupt が呼び出されます。

[指定形式]

```
void    R_RIICn_StartCondition ( 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_RIIC*n*_StopCondition

ストップ・コンディションを発行し，通信エラー／イベント発生割り込み（EEI）を発生させます。

備考 本 API 関数の呼び出しに伴い，r_riicn_error_interrupt が呼び出されます。

[指定形式]

```
void    R_RIICn_StopCondition ( 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_riic*n*_callback_receiveerror

通信エラー／イベント発生割り込み（EEI）に対応した割り込み処理のうち，アビトレーションロストの検出，NACK の検出，タイムアウトの検出に特化した処理を行います。

備考 本 API 関数は，r_riicn_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]

```
#include    "r_cg_macrodriver.h"
static  void    r_riic*n*_callback_receiveerror ( MD_STATUS *status* );
```

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	MD_STATUS *status*;	通信エラー／イベント発生割り込みの発生要因 MD_ERROR1 : アビトレーションロストの検出 MD_ERROR2 : タイムアウトの検出 MD_ERROR3 : NACK の検出

[戻り値]

なし

r_riic*n*_callback_transmitend

通信エラー／イベント発生割り込み（EEI）に対応した割り込み処理のうち，R_RIICn_Master_Send の呼び出しに伴うスタート・コンディションの検出に特化した処理を行います。

備考 本 API 関数は，r_riicn_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］

```
static  void    r_riicn_callback_transmitend ( 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_riic*n*_callback_receiveend

通信エラー／イベント発生割り込み（EEI）に対応した割り込み処理のうち，R_RIICn_Master_Receive の呼び出しに伴うスタート・コンディションの検出に特化した処理を行います。

備考 本 API 関数は，r_riicn_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]

```
static  void    r_riic*n*_callback_receiveend ( 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

3.2.28 シリアル・ペリフェラル・インタフェース（RSPI）

以下に，コード生成ツールがシリアル・ペリフェラル・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 3.28 シリアル・ペリフェラル・インタフェース用 API 関数

API 関数名	機能概要
R_RSPIn_Create	シリアル・ペリフェラル・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_RSPIn_Create_UserInit	シリアル・ペリフェラル・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_rspin_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_rspin_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_rspin_error_interrupt	RSPI エラー割り込みの発生に伴う処理を行います。
r_rspin_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_RSPIn_Start	RSPI 通信を開始します。
R_RSPIn_Stop	RSPI 通信を終了します。
R_RSPIn_Send	RSPI 送信を開始します。
R_RSPIn_Send_Receive	RSPI 送受信を開始します。
r_rspin_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_rspin_callback_error	RSPI エラー割り込みの発生に伴う処理を行います。
r_rspin_callback_transmitend	RSPI アイドル割り込みの発生に伴う処理を行います。

R_RSPI*n*_Create

シリアル・ペリフェラル・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。

[指定形式]

```
void    R_RSPIn_Create ( 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_RSPI*n*_Create_UserInit

シリアル・ペリフェラル・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_RSPIn_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_RSPIn_Create_UserInit ( 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_rspi*n*_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。

備考 本 API 関数は，受信バッファ・フル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]

```
static  void    r_rspin_receive_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_rspi*n*_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。

備考 本 API 関数は，送信バッファ・エンプティ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]

```
static  void    r_rspin_transmit_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_rspi*n*_error_interrupt

RSPI 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，RSPI 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]

```
static  void    r_rspin_error_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_rspi*n*_idle_interrupt

RSPI 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，RSPI アイドル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]

```
static  void    r_rspin_idle_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_RSPI*n*_Start

RSPI 通信を開始します。

[指定形式]

```
void    R_RSPIn_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_RSPI*n*_Stop

RSPI 通信を終了します。

[指定形式]

```
void    R_RSPIn_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_RSPI*n*_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 RSPI 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. RSPI 送信を行う際には，本 API 関数の呼び出し以前に R_RSPIn_Start を呼び出す必要があります。

[指定形式]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_RSPIn_Send ( uint8_t * const tx_buf, uint16_t tx_num );
```

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	`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_RSPI*n*_Send_Receive

RSPI 送受信を開始します。

備考 1. 本 API 関数では，RSPI 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 RSPI 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，RSPI 受信処理として，1 バイト単位の RSPI 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. RSPI 送受信を行う際には，本 API 関数の呼び出し以前に R_RSPIn_Start を呼び出す必要があります。

[指定形式]

```
#include    "r_cg_macrodriver.h"
MD_STATUS   R_RSPIn_Send_Receive ( uint8_t * const tx_buf, uint16_t tx_num, uint8_t *
const rx_buf );
```

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	`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;`	送受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

[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_rspi*n*_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_rspin_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]

```
static  void    r_rspin_callback_receiveend ( 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_rspi*n*_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 本 API 関数は，r_rspin_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]

```
static  void    r_rspin_callback_error ( uint8_t err_type );
```

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	uint8_t *err_type*;	RSPI エラー割り込みの発生要因（*x* 部は不定） *xxxx*00*x*1B : オーバラン・エラーの検出 *xxxx*01*x*0B : モード・フォルト・エラーの検出 *xxxx*10*x*0B : パリティ・エラーの検出

[戻り値]

なし

r_rspi*n*_callback_transmitend

RSPI 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_rspin_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]

```
static  void    r_rspin_callback_transmitend ( 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

3.2.29 CRC 演算器（CRC）

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 3.29 CRC 演算器用 API 関数

API 関数名	機能概要
R_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。
R_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。
R_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。
R_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。
R_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。
R_CRC_Input_Data	CRC 演算を行うデータの初期値を設定します。
R_CRC_Get_Result	演算結果を獲得します。

R_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]

RX65N/RX651 の場合

```
void    R_CRC_SetCRC8 ( void );
```

そのほかのデバイスの場合

```
#include    "r_cg_crc.h"
void    R_CRC_SetCRC8 ( crc_bitorder order );
```

[引数]

I/O	引数	説明
I	crc_bitorder *order*;	CRC 演算切り替えの種類 CRC_LSB : LSB ファースト CRC_MSB : MSB ファースト

[戻り値]

なし

R_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]

RX65N/RX651 の場合

```
void    R_CRC_SetCRC16 ( void );
```

その他のデバイスの場合

```
#include    "r_cg_crc.h"
void    R_CRC_SetCRC16 ( crc_bitorder order );
```

[引数]

I/O	引数	説明
I	crc_bitorder *order*;	CRC 演算切り替えの種類 CRC_LSB : LSB ファースト CRC_MSB : MSB ファースト

[戻り値]

なし

R_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]

RX65N/RX651 の場合

```
void    R_CRC_SetCCITT ( void );
```

その他のデバイスの場合

```
#include    "r_cg_crc.h"
void    R_CRC_SetCCITT ( crc_bitorder order );
```

[引数]

I/O	引数	説明
I	crc_bitorder *order*;	CRC 演算切り替えの種類 CRC_LSB : LSB ファースト CRC_MSB : MSB ファースト

[戻り値]

なし

R_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_CRC_SetCRC32 ( void );
```

[引数]

なし

[戻り値]

なし

R_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_CRC_SetCRC32C ( void );
```

[引数]

なし

[戻り値]

なし

R_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]

```
#include    "r_cg_macrodriver.h"
void    R_CRC_Input_Data ( uint8_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

なし

[戻り値]

なし

R_CRC_Get_Result

演算結果を獲得します。

[指定形式]

```
#include    "r_cg_macrodriver.h"
void    R_CRC_Get_Result ( uint8_t * const result );
```

[引数]

I/O	引数	説明
O	`uint8_t * const result;`	獲得した演算結果を格納する領域へのポインタ

[戻り値]

なし

3.2.30 12 ビット A/D コンバータ（S12AD）

以下に，コード生成ツールが 12 ビット A/D 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 3.30 12 ビット A/D コンバータ用 API 関数

API 関数名	機能概要
R_S12ADn_Create	12 ビット A/D 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_S12ADn_Create_UserInit	12 ビット A/D 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_s12adn_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_s12adn_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_S12ADn_Start	A/D 変換を開始します。
R_S12ADn_Stop	A/D 変換を終了します。
R_S12ADn_Get_ValueResult	変換結果を獲得します。
R_S12ADn_Set_CompareValue	コンペアレベルの設定を行います。
r_s12adn_compare_interrupt	コンペア割り込みの発生に伴う処理を行います。

R_S12AD*n*_Create

12 ビット A/D 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。

[指定形式]

```
void    R_S12ADn_Create ( 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_S12AD*n*_Create_UserInit

12 ビット A/D 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_S12ADn_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_S12AD n_Create_UserInit ( 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_s12ad*n*_interrupt

スキャン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，アナログ入力のスキャンが完了した場合に発生するスキャン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]

```
static  void    r_s12ad 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_s12ad*n*_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。

備考 本 API 関数は，グループ B に割り当てられたアナログ入力のスキャンが完了した場合に発生するグループ B スキャン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]

```
static  void    r_s12ad*n*_groupb_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_S12AD*n*_Start

A/D 変換を開始します。

[指定形式]

```
void    R_S12ADn_Start ( 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_S12AD*n*_Stop

A/D 変換を終了します。

[指定形式]

```
void    R_S12ADn_Stop ( 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_S12AD*n*_Get_ValueResult

変換結果を獲得します。

[指定形式]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_s12ad.h"
void    R_S12ADn_Get_ValueResult ( ad_channel_t channel, uint16_t * const buffer );
```

備考　　*n* は，ユニット番号を意味します。

[引数]

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 ADCHANNEL6 : 入力チャネル AN006 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 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）
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_S12AD*n*_Set_CompareValue

コンペアレベルの設定を行います。

[指定形式]

```
void    R_S12ADn_Set_CompareValue ( ad_channel_t  reg_value0,  rad_channel_t
reg_value1 );
```

備考　　*n* は，ユニット番号を意味します。

[引数]

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

[戻り値]

なし

r_s12ad*n*_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]

```
static void    r_s12adn_compare_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

3.2.31 D/A コンバータ（DA）

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 3.31 D/A コンバータ用 API 関数

API 関数名	機能概要
R_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_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_DAm_Start	D/A 変換を開始します。
R_DAm_Stop	D/A 変換を終了します。
R_DAm_Set_ConversionValue	D/A 変換を行うデータを設定します。

R_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。

[指定形式]

```
void    R_DA_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_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_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_DA_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

R_DA*m*_Start

D/A 変換を開始します。

[指定形式]

```
void    R_DAm_Start ( void );
```

備考　　*m* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_DA*m*_Stop

D/A 変換を終了します。

[指定形式]

```
void    R_DAm_Stop ( void );
```

備考　　*m* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_DA*m*_Set_ConversionValue

D/A 変換を行うデータを設定します。

[指定形式]

```
#include    "r_cg_macrodriver.h"
void    R_DAm_Set_ConversionValue ( uint16_t reg_value );
```

備考　　*m* 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

3.2.32 12 ビット D/A コンバータ（R12DA）

以下に，コード生成ツールが 12 ビット 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 3.32 12 ビット D/A コンバータ用 API 関数

API 関数名	機能概要
R_R12DA_Create	12 ビット D/A コンバータの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_R12DAn_Start	D/A 変換を開始します。
R_R12DAn_Stop	D/A 変換を終了します。
R_R12DAn_Set_ConversionValue	D/A 変換するデータを格納します。
R_R12DA_Sync_Start	D/A 一括変換を開始します。
R_R12DA_Sync_Stop	D/A 一括変換を終了します。
R_R12DA_Create_UserInit	12 ビット 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_R12DA_Create

12 ビット D/A コンバータの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。

[指定形式]

```
void    R_R12DA_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_R12DA*n*_Start

D/A 変換を開始します。

[指定形式]

```
void    R_R12DAn_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_R12DA*n*_Stop

D/A 変換を終了します。

[指定形式]

```
void    R_R12DAn_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_R12DA*n*_Set_ConversionValue

D/A 変換するデータを格納します。

[指定形式]

```
void    R_R12DAn_Set_ConversionValue ( uint16 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 変換値

[戻り値]

なし

R_R12DA_Sync_Start

D/A 一括変換を開始します。

[指定形式]

```
void    R_R12DA_Sync_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_R12DA_Sync_Stop

D/A 一括変換を終了します。

[指定形式]

```
void    R_R12DA_Sync_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_R12DA_Create_UserInit

12 ビット 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_R12DA_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_R12DA_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

3.2.33 コンパレータ B（CMPB）

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

表 3.33 コンパレータ B 用 API 関数

API 関数名	機能概要
R_CMPB_Create	コンパレータ B を制御するうえで必要となる初期化処理を行います。
R_CMPB_Create_UserInit	コンパレータ B に関するユーザ独自の初期化処理を行います。
r_cmpb_cmpbn_interrupt	コンパレータ Bn 割り込みの発生に伴う処理を行います。
R_CMPBn_Start	アナログ入力電圧の比較を開始します。
R_CMPBn_Stop	アナログ入力電圧の比較を終了します。

R_CMPB_Create

コンパレータ B を制御するうえで必要となる初期化処理を行います。

[指定形式]

```
void    R_CMPB_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_CMPB_Create_UserInit

コンパレータ B 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_CMPB_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_CMPB_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_cmpb_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]

```
static  void    r_cmpb_cmpbn_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_CMPB*n*_Start

アナログ入力電圧の比較を開始します。

[指定形式]

```
void    R_CMPBn_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_CMPB*n*_Stop

アナログ入力電圧の比較を終了します。

[指定形式]

```
void    R_CMPBn_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

3.2.34 データ演算回路（DOC）

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 3.34 データ演算回路用 API 関数

API 関数名	機能概要
R_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_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_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。
R_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_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_DOC_GetResult	演算結果を獲得します。
R_DOC_ClearFlag	データ演算回路フラグをクリアします。

R_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。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_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_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_DOC_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_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。

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]

```
static  void    r_doc_dopcf_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_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。

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]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_doc.h"
void    R_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*;	データ演算を行うデータの初期値

[戻り値]

なし

R_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]

```
#include    "r_cg_macrodriver.h"
void    R_DOC_WriteData ( uint16_t data );
```

[引数]

I/O	引数	説明
I	uint16_t *data*;	データ演算を行う値

[戻り値]

なし

R_DOC_GetResult

演算結果を獲得します。

[指定形式]

```
#include    "r_cg_macrodriver.h"
void    R_DOC_GetResult ( uint16_t * const data );
```

[引数]

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

[戻り値]

なし

R_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_DOC_ClearFlag ( void );
```

[引数]

なし

[戻り値]

なし

3.2.35 ローパワータイマ（LPT）

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 3.35 ローパワータイマ用 API 関数

API 関数名	機能概要
R_LPT_Create	ローパワータイマの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_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_LPT_Start	ローパワータイマのカウントを開始します。
R_LPT_Stop	ローパワータイマのカウントを終了します。

R_LPT_Create

ローパワータイマの機能を制御する上で必要となる初期化処理を行います。

[指定形式]

```
void    R_LPT_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_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_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_LPT_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

R_LPT_Start

ローパワータイマのカウントを開始します。

[指定形式]

```
void    R_LPT_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_LPT_Stop

ローパワータイマのカウントを終了します。

[指定形式]

```
void    R_LPT_Stop ( void );
```

[引数]

なし

[戻り値]

なし

3.2.36 コンパレータ C（CMPC）

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

表 3.36 コンパレータ C 用 API 関数

API 関数名	機能概要
R_CMPC_Create	コンパレータ C を制御するうえで必要となる初期化処理を行います。
R_CMPC_Create_UserInit	コンパレータ C に関するユーザ独自の初期化処理を行います。
r_cmpc_cmpcn_interrupt	コンパレータ Cn 割り込みの発生に伴う処理を行います。
R_CMPCn_Start	アナログ入力電圧の比較を開始します。
R_CMPCn_Stop	アナログ入力電圧の比較を終了します。

R_CMPC_Create

コンパレータ C を制御するうえで必要となる初期化処理を行います。

[指定形式]

```
void    R_CMPC_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_CMPC_Create_UserInit

コンパレータ C 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_CMPC_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_CMPC_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_cmpc_cmpc*n*_interrupt

コンパレータ C*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 C*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]

```
static  void    r_cmpc_cmpcn_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_CMPC*n*_Start

アナログ入力電圧の比較を開始します。

[指定形式]

```
void    R_CMPCn_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_CMPC*n*_Stop

アナログ入力電圧の比較を終了します。

[指定形式]

```
void    R_CMPCn_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

3.2.37 LCD コントローラ / ドライバ（LCD）

以下に，コード生成ツールが LCD コントローラ / ドライバ用として出力する API 関数の一覧を示します。

表 3.37 LCD コントローラ / ドライバ用 API 関数

API 関数名	機能概要
R_LCD_Create	LCD コントローラ / ドライバを制御するうえで必要となる初期化処理を行います。
R_LCD_Create_UserInit	LCD コントローラ / ドライバに関するユーザ独自の初期化処理を行います。
R_LCD_Start	LCD コントローラ / ドライバを表示オン状態にします。
R_LCD_Stop	LCD コントローラ / ドライバを表示オフ状態にします。
R_LCD_Voltage_On	内部昇圧回路 , および容量分割回路を動作可能状態にします。
R_LCD_Voltage_Off	内部昇圧回路 , および容量分割回路を動作停止状態にします。

R_LCD_Create

LCD コントローラ / ドライバを制御するうえで必要となる初期化処理を行います。

[指定形式]

```
void    R_LCD_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_LCD_Create_UserInit

LCD コントローラ / ドライバ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_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_LCD_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

R_LCD_Start

LCD コントローラ / ドライバを表示オン状態にします。

[指定形式]

```
void    R_LCD_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_LCD_Stop

LCD コントローラ / ドライバを表示オフ状態にします。

[指定形式]

```
void    R_LCD_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_LCD_Voltage_On

内部昇圧回路 , および容量分割回路を動作可能状態にします。

[指定形式]

```
void    R_LCD_Voltage_On ( void );
```

[引数]

なし

[戻り値]

なし

R_LCD_Voltage_Off

内部昇圧回路 , および容量分割回路を動作停止状態にします。

[指定形式]

```
void    R_LCD_Voltage_Off ( 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	2014.08.01	―	初版発行
1.10	2014.06.27	全体	RX64M 対応
1.20	2014.08.01	全体	コンパレータ B 追加
		全体	R_LPC_AllModuleStop に変更
		7 ～ 15	表 2.1 出力ファイル 変更
		3.2.8 DMA コントローラに変更	
		92	DMA コントローラに変更
		3.2.21 コンペア・マッチ・タイマ W（CMTW）	
		175	アウトプット・コンペアに変更
		3.2.25 シリアル・コミュニケーション・インタフェース（SCI）	
		217 ～ 243	r_scin_receive_interrupt に変更
			スタート・コンディションに変更
			ストップ・コンディションに変更
		3.2.26 FIFO 内蔵シリアル・コミュニケーション・インタフェース（SCIFA）	
		244	表 3.26 FIFO 内蔵シリアル・コミュニケーション・インタフェース用 API 関数 変更
1.30	2016.10.01	全体	RX65N / RX651 対応
		全体	ローパワータイマ 追加
		全体	コンパレータ C 追加
		全体	LCD コントローラ / ドライバ 追加
		7 ～ 16	表 2.1 出力ファイル 変更
		3.2.1 共通	
		22	PowerON_Reset_PC を追加
		23	r_privileged_exception を追加
		24	r_floatingpoint_exception を追加
		25	r_access_exception を追加
		29	備考の誤記を訂正
		3.2.4 クロック周波数精度測定回路（CAC）	
		51	r_cac_ovff_interrupt に変更
		3.2.9 データ・トランスファ・コントローラ（DTC）	
		99	備考に説明を追加
		100	備考に説明を追加

Rev.	発行日	改訂内容	
		ページ	ポイント
1.30	2016.10.01	3.2.10 イベント・リンク・コントローラ（ELC）	
		104	備考に説明を追加
		3.2.12 マルチファンクション・タイマ・パルス・ユニット（MTU2）	
		117	r_mtu2_cj_tgimn_interrupt を追加
		119	r_mtu2_cj_tcivn_interrupt を追加
		3.2.13 マルチファンクション・タイマ・パルス・ユニット（MTU3）	
		127	r_mtu3_cj_tgimn_interrupt を追加
		129	r_mtu3_cj_tcivn_interrupt を追加
		3.2.14 ポート・アウトプット・イネーブル（POE2）	
		139	R_POE2_Set_HiZ_MTUn を追加
		140	R_POE2_Clear_HiZ_MTUn を追加
		3.2.14 ポート・アウトプット・イネーブル（POE3）	
		147	R_POE3_Set_HIZ_MTUn を追加
		148	R_POE3_Clear_HIZ_MTUn を追加
		149	R_POE3_Set_HiZ_GPTn を追加
		150	R_POE3_Clear_HiZ_GPTn を追加
		3.2.23 ウォッチドッグ・タイマ（WDT）	
		224	r_wdt_nmi_interrupt を追加
		3.2.29 CRC 演算器（CRC）	
		311	R_CRC_SetCRC32 を追加
		312	R_CRC_SetCRC32C を追加

CS+ コード生成ツール　ユーザーズマニュアル
RX API リファレンス編

発行年月日　2014年 8月 1日　Rev.1.00
　　2016年10月 1日　Rev.1.30

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)

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口

http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

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）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：http://japan.renesas.com/contact/

Colophon 3.0

CS+ コード生成ツール

ルネサス エレクトロニクス株式会社

R20UT3103JJ0130