User's Manual

コード生成ツール

ユーザーズマニュアル　RL78 API リファレンス編

対象デバイス
RL78 ファミリ

ルネサス エレクトロニクス
www.renesas.com

Rev.1.05　2024.7

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。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、お客様の責任において、お客様の機器・システムを設計ください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品または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またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 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. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング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。
6. 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）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等

   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、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（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合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。
9. 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、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他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。
10. 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。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。
11. 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。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
12. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
13. 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。
14. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

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直接的、間接的に支配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。

(Rev.5.0-1　2020.10)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

このマニュアルの使い方

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

目的　　このマニュアルは、コード生成ツールのドライバコード生成の持つソフトウェ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ェア、ソフトウェア開発の参照用資料として役立つことを目的としています。

構成　　このマニュアルは、大きく分けて次の内容で構成しています。

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

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

目次

1. 概　説

コード生成ツールは、デバイス・ドライバを自動生成するソフトウェア・ツールです。
このドキュメントでは、コード生成ツールが出力するファイルおよび API 関数について説明します。

1.1 概　要

コード生成ツールは、GUI ベースで各種情報を設定することにより、マイクロコントローラの端子配置状況（端子配置表，端子配置図）／マイクロコントローラが提供している周辺機能（クロック発生回路，ポート機能など）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。
コード生成ツールは、Windows オペレーティングシステムをサポートするだけでなく、e^2 studio 2024-07 以降、Linux および Mac OS もサポートします。

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
  コード生成ツールでは、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，リンク・ディレクティブ・ファイル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
  端子配置／コード生成ツール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 ※1
  コード生成ツール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。

※1 Windows 版コード生成ツールのみ、リネーム機能に対応しています。Linux 版、Mac OS 版では、本機能には対応していません。

1.3　対応コンパイラ

コード生成ツールで生成されるコードは、以下のコンパイラでビルド可能です。

Windows 版
- ルネサス製コンパイラ(CC-RL，CA78K0R)
- GNU コンパイラ
- IAR 製コンパイラ
- LLVM コンパイラ

Linux 版 ※2
- ルネサス製コンパイラ(CC-RL)
- LLVM コンパイラ

Mac OS 版 ※2
- LLVM コンパイラ

※2 Linux と Mac OS は、e^2 studio 2024-07 からサポートされました。

1.4　注　　意

以下、コード生成ツールを使用する上での注意事項を説明します。

- OSS（Open Source Software）について
  コード生成ツールは、OSS を使用しておりません。

- 多重割り込みについて
  コード生成ツールは、多重割り込みに対応していません。ご使用のコンパイラマニュアルを参照し、コードを書き換えてください。

- グローバル変数の使用について
  各周辺機能の Create()関数内でグローバル変数を初期化しても、スタートアップ内の RAM 初期化でクリアされるため、main()関数の実行時にグローバル変数はクリアされています。(ルネサス製コンパイラ、IAR 製コンパイラ用のコード生成時)

2. 出力ファイル

本章では、コード生成ツール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、コード生成ツールが出力するファイルの一覧を示します。

表 2.1　出力ファイル(1/15)

周辺機能	ファイル名	API 関数名	出力 (*1)
共通	r_main.c または r_cg_main.c	main R_MAIN_UserserInit	○ ○
	r_systeminit.c または r_cg_sysmteminit.c	hdwinit R_Systeminit __low_level_init (*3)	○ ○ ○
	r_cg_macrodriver.h	－	－
	r_cg_userdefine.h	－	－
	r_reset_program.asm または r_cg_reset_program.asm (*2) または start.S (*4)	－	－
	r_hardware_setup.c または r_cg_hardware_setup.c (*2, *4)	R_Systeminit HardwareSetup	○ ○
	r_cg_vector_table.c (*2, *4)	－	－
	r_cg_interrupt_handlers.h (*2, *4)	－	－
	r_cg_config.h	－	－
	r_cg_inthandler.c (*4)	－	－
クロック発生回路	r_cg_cgc.c	R_CGC_Create R_CGC_Set_ClockMode R_CGC_RAMECC_Start R_CGC_RAMECC_Stop R_CGC_StackPointer_Start R_CGC_StackPointer_Stop R_CGC_ClockMonitor_Start R_CGC_ClockMonitor_Stop	○ × ○ ○ ○ ○ ○ ○
	r_cg_cgc_user.c	R_CGC_Create_UserInit r_cgc_ram_ecc_interrupt r_cgc_stackpointer_interrupt r_cgc_clockmonitor_interrupt R_CGC_Get_ResetSource	× ○ ○ ○ ○
	r_cg_cgc.h	－	－
ポート機能	r_cg_port.c	R_PORT_Create	○
	r_cg_port_user.c	R_PORT_Create_UserInit	×
	r_cg_port.h	－	－
高速オンチップ・オシレータ・クロック周波数補正機能	r_cg_hofc.c	R_HOFC_Create R_HOFC_Start R_HOFC_Stop	○ ○ ○
	r_cg_hofc_user.c	R_HOFC_Create_UserInit r_hofc_interrupt	× ○
	r_cg_hofc.h	－	－

表 2.2　出力ファイル(2/15)

周辺機能	ファイル名	API 関数名	出力 (*1)
タイマ・アレイ・ユニット	r_cg_timer.c または r_cg_tau.c	R_TAUm_Create R_TAUm_Channeln_Start R_TAUm_Channeln_Higher8bits_Start R_TAUm_Channeln_Lower8bits_Start R_TAUm_Channeln_Stop R_TAUm_Channeln_Higher8bits_Stop R_TAUm_Channeln_Lower8bits_Stop R_TAUm_Reset R_TAUm_Set_PowerOff R_TAUm_Channeln_Get_PulseWidth R_TAUm_Channeln_Set_SoftwareTriggerOn	○ ○ ○ ○ ○ ○ ○ × ○ ○ ○
	r_cg_timer_user.c または r_cg_tau_user.c	R_TAUm_Create_UserInit r_taum_channeln_interrupt r_taum_channeln_higher8bits_interrupt	× ○ ○
	r_cg_timer.h または r_cg_tau.h	－	－
タイマ RJ	r_cg_timer.c または r_cg_tmrj.c	R_TMR_RJ0_Create R_TMR_RJ0_Start R_TMR_RJ0_Stop R_TMR_RJ0_Set_PowerOff R_TMR_RJ0_Get_PulseWidth R_TMRJ0_Create R_TMRJ0_Start R_TMRJ0_Stop R_TMRJ0_Set_PowerOff R_TMRJ0_Get_PulseWidth	○ ○ ○ ○ ○ ○ ○ ○ × ×
	r_cg_timer_user.c または r_cg_tmrj_user.c	R_TMR_RJ0_Create_UserInit r_tmr_rj0_interrupt R_TMRJ0_Create_UserInit r_tmrj0_interrupt	× ○ × ○
	r_cg_timer.h または r_cg_tmrj.h	－	－

表 2.3　出力ファイル(3/15)

周辺機能	ファイル名	API 関数名	出力 (*1)
タイマ RD	r_cg_timer.c または r_cg_tmrd.c	R_TMR_RDn_Create	○
		R_TMR_RDn_Start	○
		R_TMR_RDn_Stop	○
		R_TMR_RDn_Set_PowerOff	×
		R_TMR_RDn_ForcedOutput_Start	×
		R_TMR_RDn_ForcedOutput_Stop	×
		R_TMR_RDn_Get_PulseWidth	○
		R_TMRDn_Create	×
		R_TMRDn_Start	×
		R_TMRDn_Stop	×
		R_TMRDn_Set_PowerOff	○
		R_TMRDn_ForcedOutput_Start	○
		R_TMRDn_ForcedOutput_Stop	○
		R_TMRDn_Get_PulseWidth	○
		R_TMRD_Set_PowerOff	×
		R_TMRD_PWMOP_ForcedOutput_Stop	○
		R_TMRD_PWMOP_Set_PowerOff	○
	r_cg_timer_user.c または r_cg_tmrd_user.c	R_TMR_RDn_Create_UserInit	×
		r_tmr_rdn_interrupt	○
		R_TMRDn_Create_UserInit	×
		r_tmrdn_interrupt	○
	r_cg_timer.h または r_cg_tmrd.h	－	－
タイマ RG	r_cg_timer.c	R_TMR_RG0_Create	○
		R_TMR_RG0_Start	○
		R_TMR_RG0_Stop	○
		R_TMR_RG0_Set_PowerOff	×
		R_TMR_RG0_Get_PulseWidth	○
	r_cg_timer_user.c	R_TMR_RG0_Create_UserInit	×
		r_tmr_rg0_interrupt	○
	r_cg_timer.h	－	－
タイマ RX	r_cg_tmrx.c	R_TMRX_Create	○
		R_TMRX_Start	○
		R_TMRX_Stop	○
		R_TMRX_Set_PowerOff	○
		R_TMRX_Get_BufferValue	×
	r_cg_tmrx_user.c	R_TMRX_Create_UserInit	×
		r_tmrx_interrupt	○
	r_cg_tmrx.h	－	－
16 ビット・タイマ KB	r_cg_timer.c または r_cg_tmkb.c	R_TMR_KB_Create	○
		R_TMR_KBm_Start	○
		R_TMR_KBm_Stop	○
		R_TMR_KBm_Set_PowerOff	×
		R_TMR_KBmn_ForcedOutput_Start	○
		R_TMR_KBmn_ForcedOutput_Stop	○
		R_TMR_KBm_BatchOverwriteRequestOn	○
		R_TMR_KBm_ForcedOutput_mn_Start	○
		R_TMR_KBm_ForcedOutput_mn_Stop	○
		R_TMR_KBm_Reset	×
	r_cg_timer_user.c または r_cg_tmkb_user.c	R_TMR_KB_Create_UserInit	×
		r_tmr_kbm_interrupt	○
	r_cg_timer.h または r_cg_tmkb.h	－	－

表 2.4　出力ファイル(4/15)

周辺機能	ファイル名	API 関数名	出力 (*1)
16 ビット・タイマ KC0	r_cg_timer.c	R_TMR_KC0_Create	○
		R_TMR_KC0_Start	○
		R_TMR_KC0_Stop	○
		R_TMR_KC0_Set_PowerOff	×
	r_cg_timer_user.c	R_TMR_KC0_Create_UserInit	×
		r_tmr_kc0_interrupt	○
	r_cg_timer.h	－	－
16 ビット・タイマ KB2	r_cg_kb2.c	R_KB2m_Create	○
		R_KB2m_Start	○
		R_KB2m_Stop	○
		R_KB2m_Set_PowerOff	×
		R_KB2m_Simultaneous_Start	○
		R_KB2m_Simultaneous_Stop	○
		R_KB2m_Synchronous_Start	○
		R_KB2m_Synchronous_Stop	○
		R_KB2m_TKBOn0_Forced_Output_Stop_Function1_Start	○
		R_KB2m_TKBOn0_Forced_Output_Stop_Function1_Stop	○
		R_KB2m_TKBOn1_Forced_Output_Stop_Function1_Start	○
		R_KB2m_TKBOn1_Forced_Output_Stop_Function1_Stop	○
		R_KB2m_TKBOn0_DitheringFunction_Start	○
		R_KB2m_TKBOn0_DitheringFunction_Stop	○
		R_KB2m_TKBOn1_DitheringFunction_Start	○
		R_KB2m_TKBOn1_DitheringFunction_Stop	○
		R_KB2m_TKBOn0_SmoothStartFunction_Start	○
		R_KB2m_TKBOn0_SmoothStartFunction_Stop	○
		R_KB2m_TKBOn1_SmoothStartFunction_Start	○
		R_KB2m_TKBOn1_SmoothStartFunction_Stop	○
		R_KB2m_BatchOverwriteRequestOn	○
	r_cg_kb2_user.c	R_KB2m_Create_UserInit	×
		r_kb2m_interrupt	○
	r_cg_kb2.h	－	－

表 2.5　出力ファイル(5/15)

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	r_cg_rtc.c	R_RTC_Create	○
		R_RTC_Start	○
		R_RTC_Stop	○
		R_RTC_Set_PowerOff	○
		R_RTC_Set_HourSystem	×
		R_RTC_Set_CounterValue	○
		R_RTC_Set_CalendarCounterValue	○
		R_RTC_Set_BinaryCounterValue	○
		R_RTC_Get_CounterValue	○
		R_RTC_Get_CalendarCounterValue	○
		R_RTC_Get_BinaryCounterValue	○
		R_RTC_Set_ConstPeriodInterruptOn	○
		R_RTC_Set_ConstPeriodInterruptOff	○
		R_RTC_Set_AlarmOn	○
		R_RTC_Set_CalendarAlarmOn	○
		R_RTC_Set_BinaryAlarmOn	○
		R_RTC_Set_AlarmOff	○
		R_RTC_Set_AlarmValue	○
		R_RTC_Set_CalenderAlarmValue	○
		R_RTC_Set_BinaryAlarmValue	○
		R_RTC_Get_AlarmValue	○
		R_RTC_Get_CalenderAlarmValue	○
		R_RTC_Get_BinaryAlarmValue	○
		R_RTC_Set_RTC1HZOn	○
		R_RTC_Set_RTC1HZOff	○
		R_RTC_Set_RTCOUTOn	○
		R_RTC_Set_RTCOUTOff	○
	r_cg_rtc_user.c	R_RTC_Create_UserInit	×
		r_rtc_interrupt	○
		r_rtc_callback_constperiod	○
		r_rtc_callback_alarm	○
		r_rtc_alarminterrupt	○
		r_rtc_periodinterrupt	○
		r_rtc_callback_periodic	○
	r_cg_rtc.h	―	―
サブシステム・クロック周波数測定回路	r_cg_fmc.c	R_FMC_Create	○
		R_FMC_Start	○
		R_FMC_Stop	○
		R_FMC_Set_PowerOff	×
	r_cg_fmc_user.c	R_FMC_Create_UserInit	×
		r_fmc_interrupt	○
	r_cg_fmc.h	―	―

表 2.6　出力ファイル(6/15)

周辺機能	ファイル名	API 関数名	出力 (*1)
12 ビット・インターバル・タイマ	r_cg_it.c	R_IT_Create R_IT_Start R_IT_Stop R_IT_Reset R_IT_Set_PowerOff	○ ○ ○ × ×
	r_cg_it_user.c	R_IT_Create_UserInit r_it_interrupt	× ○
	r_cg_it.h	－	－
8 ビット・インターバル・タイマ	r_cg_it8bit.c	R_IT8Bitm_Channeln_Create R_IT8Bitm_Channeln_Start R_IT8Bitm_Channeln_Stop R_IT8Bitm_Channeln_Set_PowerOff R_IT8Bitm_Set_PowerOff	○ ○ ○ × ×
	r_cg_it8bit_user.c	R_IT8Bitm_Channeln_Create_UserInit r_it8bitm_channeln_interrupt	× ○
	r_cg_it8bit.h	－	－
16 ビット・ウェイクアップ・タイマ	r_cg_timer.c	R_WUTM_Create R_WUTM_Start R_WUTM_Stop R_WUTM_Set_PowerOff	○ ○ ○ ×
	r_cg_timer_user.c	R_WUTM_Create_UserInit r_wutm_interrupt	× ○
	r_cg_timer.h	－	－
クロック出力／ブザー出力制御回路	r_cg_pclbuz.c	R_PCLBUZn_Create R_PCLBUZn_Start R_PCLBUZn_Stop R_PCLBUZn_Set_PowerOff	○ ○ ○ ×
	r_cg_pclbuz_user.c	R_PCLBUZn_Create_UserInit	×
	r_cg_gpt.h	－	－
ウォッチドッグ・タイマ	r_cg_wdt.c	R_WDT_Create R_WDT_Restart	○ ○
	r_cg_wdt_user.c	R_WDT_Create_UserInit r_wdt_interrupt	× ○
	r_cg_wdt.h	－	－
プログラマブル・ゲイン係数アンプ付き 24 ビット ΔΣ コンバータ	r_cg_pga_dsad.c	R_PGA_DSAD_Create R_PGA_DSAD_Start R_PGA_DSAD_Stop R_PGA_DSAD_Set_PowerOff R_PGA_DSAD_Get_AverageResult R_PGA_DSAD_Get_Result R_PGA_DSAD_CAMP_OffsetTrimming	○ ○ ○ × ○ ○ ○
	r_cg_pga_dsad_user.c	R_PGA_DSAD_Create_UserInit r_pga_dsad_interrupt_conversion r_pga_dsad_interrupt_scan r_pga_dsad_conversion_interrupt r_pga_dsad_scan_interrupt	× ○ ○ ○ ○
	r_cg_pga_dsad.h	－	－

表 2.7　出力ファイル(7/15)

周辺機能	ファイル名	API 関数名	出力 (*1)
A/D コンバータ	r_cg_adc.c	R_ADC_Create	○
		R_ADC_Set_OperationOn	○
		R_ADC_Set_OperationOff	○
		R_ADC_Start	○
		R_ADC_Stop	○
		R_ADC_Reset	×
		R_ADC_Set_PowerOff	×
		R_ADC_Set_ADChannel	×
		R_ADC_Set_SnoozeOn	×
		R_ADC_Set_SnoozeOff	×
		R_ADC_Set_TestChannel	×
		R_ADC_Get_Result	○
		R_ADC_Get_Result_8bit	○
	r_cg_adc_user.c	R_ADC_Create_UserInit	×
		r_adc_interrupt	○
	r_cg_adc.h	－	－
コンフィギュラブル・アンプ	r_cg_camp.c	R_CAMP_Create	○
		R_CAMPn_Start	○
		R_CAMPn_Stop	○
		R_CAMP_Set_PowerOff	×
	r_cg_camp_user.c	R_CAMP_Create_UserInit	×
	r_cg_camp.h	－	－
温度センサ	r_cg_tmps.c	R_TMPS_Create	○
		R_TMPS_Start	○
		R_TMPS_Stop	○
		R_TMPS_Reset	×
		R_TMPS_Set_PowerOff	×
	r_cg_tmps_user.c	R_TMPS_Create_UserInit	×
	r_cg_tmps.h	－	－
24 ビット ΔΣA/D コンバータ	r_cg_dsadci.c	R_DSADC_Create	○
		R_DSADC_Set_OperationOn	○
		R_DSADC_Set_OperationOff	○
		R_DSADC_Start	○
		R_DSADC_Stop	○
		R_DSADC_Reset	×
		R_DSADC_Set_PowerOff	×
		R_DSADC_Channeln_Get_Result	○
		R_DSADC_Channeln_Get_Result_16bit	○
	r_cg_dsadc_user.c	R_DSADC_Create_UserInit	×
		r_dsadc_interrupt	○
		r_dsadzcn_interrupt	○
	r_cg_dsadc.h	－	－

表 2.8　出力ファイル(8/15)

周辺機能	ファイル名	API 関数名	出力 (*1)
D/A コンバータ	r_cg_dac.c	R_DAC_Create	○
		R_DACn_Start	○
		R_DACn_Stop	○
		R_DAC_Set_PowerOff	×
		R_DACn_Set_ConversionValue	○
		R_DACn_Change_OutputVoltage_8bit	○
		R_DACn_Change_OutputVoltage	○
		R_DACn_Create	○
		R_DAC_Reset	×
	r_cg_dac_user.c	R_DAC_Create_UserInit	×
	r_cg_dac.h	－	－
プログラマブル・ゲイン・アンプ	r_cg_pga.c	R_PGA_Create	○
		R_PGA_Start	○
		R_PGA_Stop	○
		R_PGA_Reset	×
		R_PGA_Set_PowerOff	×
	r_pga_user.c	R_PGA_Create_UserInit	×
	r_pga.h	－	－
コンパレータ	r_cg_comp.c	R_COMP_Create	○
		R_COMPn_Start	○
		R_COMPn_Stop	○
		R_COMP_Reset	×
		R_COMP_Set_PowerOff	×
	r_cg_comp_user.c	R_COMP_Create_UserInit	×
		r_compn_interrupt	○
	r_cg_comp.h	－	－
コンパレータ／プログラマブル・ゲイン・アンプ	r_cg_comppgacmpb.c	R_COMPPGA_Create	○
		R_COMPPGA_Set_PowerOff	×
		R_COMPn_Start	○
		R_COMPn_Stop	○
		R_PGA_Start	○
		R_PGA_Stop	○
		R_PWMOPT_Start	○
		R_PWMOPT_Stop	○
	r_cg_comppga_user.c	R_COMPPGA_Create_UserInit	×
		r_compn_interrupt	○
	r_cg_comppga.h	－	－

表 2.9　出力ファイル(9/15)

周辺機能	ファイル名	API 関数名	出力 (*1)
シリアル・アレイ・ユニット	r_cg_serial.c または r_cg_sau.c	R_SAUm_Create	○
		R_SAUm_Reset	×
		R_SAUm_Set_PowerOff	×
		R_SAUm_Set_SnoozeOn	×
		R_SAUm_Set_SnoozeOff	×
		R_UARTn_Create	○
		R_UARTn_Start	○
		R_UARTn_Stop	○
		R_UARTn_Send	○
		R_UARTn_Receive	○
		R_CSImn_Create	○
		R_CSImn_Start	○
		R_CSImn_Stop	○
		R_CSImn_Send	○
		R_CSImn_Receive	○
		R_CSImn_Send_Receive	○
		R_IICmn_Create	○
		R_IICmn_StartCondition	○
		R_IICmn_StopCondition	○
		R_IICmn_Stop	○
		R_IICmn_Master_Send	○
		R_IICmn_Master_Receive	○
	r_cg_serial_user.c または r_cg_sau_user.c	R_SAUm_Create_UserInit	×
		r_uartn_interrupt_send	○
		r_uartn_interrupt_receive	○
		r_uartn_interrupt_error	○
		r_uartn_callback_sendend	○
		r_uartn_callback_receiveend	○
		r_uartn_callback_error	○
		r_uartn_callback_softwareoverrun	○
		r_csimn_interrupt	○
		r_csimn_callback_sendend	○
		r_csimn_callback_receiveend	○
		r_csimn_callback_error	○
		r_iicmn_interrupt	○
		r_iicmn_callback_master_sendend	○
		r_iicmn_callback_master_receiveend	○
		r_iicmn_callback_master_error	○
	r_cg_serial.h または r_cg_sau.h	－	－
シリアル・アレイ・ユニット 4	r_cg_serial.c	R_DALIn_Create	○
		R_DALIn_Start	○
		R_DALIn_Stop	○
		R_DALIn_Send	○
		R_DALIn_Receive	○
	r_cg_serial_user.c	r_dalin_interrupt_send	○
		r_dalin_interrupt_receive	○
		r_dalin_interrupt_error	○
		r_dalin_callback_sendend	○
		r_dalin_callback_receiveend	○
		r_dalin_callback_error	○
		r_dalin_callback_softwareoverrun	○
	r_cg_serial.h	－	－

表 2.10　出力ファイル(10/15)

周辺機能	ファイル名	API 関数名	出力 (*1)
アシンクロナス・シリアル・インタフェース LIN-UART	r_cg_serial.c	R_UARTFn_Create	○
		R_UARTFn_Start	○
		R_UARTFn_Stop	○
		R_UARTFn_Set_PowerOff	
		R_UARTFn_Send	○
		R_UARTFn_Receive	○
		R_UARTFn_Set_DataComparisonOn	○
		R_UARTFn_Set_DataComparisonOff	○
	r_cg_serial_user.c	R_UARTFn_Create_UserInit	×
		r_uartfn_interrupt_send	○
		r_uartfn_interrupt_receive	○
		r_uartfn_interrupt_error	○
		r_uartfn_callback_sendend	○
		r_uartfn_callback_receiveend	○
		r_uartfn_callback_error	○
		r_uartfn_callback_softwareoverrun	○
		r_uartfn_callback_expbitdetect	○
		r_uartfn_callback_idmatch	○
	r_cg_serial.h	－	－
シリアル・インタフェース IICA	r_cg_serial.c または r_cg_iica.c	R_IICAn_Create	○
		R_IICAn_StopCondition	×
		R_IICAn_Stop	○
		R_IICAn_Reset	×
		R_IICAn_Set_PowerOff	×
		R_IICAn_Master_Send	○
		R_IICAn_Master_Receive	○
		R_IICAn_Slave_Send	○
		R_IICAn_Slave_Receive	○
		R_IICAn_Set_SnoozeOn	○
		R_IICAn_Set_SnoozeOff	○
		R_IICAn_Set_WakeupOn	○
		R_IICAn_Set_WakeupOff	○
	r_cg_serila_user.c または r_cg_iica_user.c	R_IICAn_Create_UserInit	×
		r_iican_interrupt	○
		r_iican_callback_master_sendend	○
		r_iican_callback_master_receiveend	○
		r_iican_callback_master_error	○
		r_iican_callback_slave_sendend	○
		r_iican_callback_slave_receiveend	○
		r_iican_callback_slave_error	○
		r_iican_callback_getstopcondition	×
	r_cg_serial.h または r_cg_iica.h	－	－
LCD コントローラ／ドライバ	r_cg_lcd.c	R_LCD_Create	○
		R_LCD_Start	○
		R_LCD_Stop	○
		R_LCD_Set_VoltageOn	○
		R_LCD_Set_VoltageOff	○
		R_LCD_Set_PowerOff	○
		R_LCD_VoltageOn	○
		R_LCD_VoltageOff	○
	r_cg_lcd_user.c	R_LCD_Create_UserInit	×
		r_lcd_interrupt	○
	r_cg_lcd.h	－	－

表 2.11　出力ファイル(11/15)

周辺機能	ファイル名	API 関数名	出力 (*1)
サウンド・ジェネレータ	r_cg_sg.c	R_SG_Create R_SG_Start R_SG_Stop	○ ○ ○
	r_cg_sg_user.c	R_SG_Create_UserInit r_sg_interrupt	× ○
	r_cg_sg.h	－	－
DMA コントローラ	r_cg_dmac.c	R_DMACn_Create R_DMAC_Create R_DMACn_Start R_DMACn_Stop R_DMACn_Set_SoftwareTriggerOn	○ ○ ○ ○ ○
	r_cg_dmac_user.c	R_DMACn_Create_UserInit R_DMAC_Create_UserInit r_dmacn_interrupt	× × ○
	r_cg_dmac.h	－	－
データ・トランスファ・コントローラ	r_cg_dtc.c	R_DTC_Create R_DTCn_Start R_DTCn_Stop R_DTC_Set_PowerOff R_DTCDn_Start R_DTCDn_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	r_cg_elc.c	R_ELC_Create R_ELC_Stop	○ ○
	r_cg_elc_user.c	R_ELC_Create_UserInit	×
	r_cg_elc.h	－	－
割り込み機能	r_cg_intc.c	R_INTC_Create R_INTCn_Start R_INTCn_Stop R_INTCLRn_Start R_INTCLRn_Stop R_INTRTCICn_Start R_INTRTCICn_Stop R_INTFO_Start R_INTFO_Stop R_INTFO_ClearFlag	○ ○ ○ ○ ○ ○ ○ ○ ○ ○
	r_cg_intc_user.c	R_KEY_Create_UserInit r_intcn_interrupt r_intclrn_interrupt r_intrtcicn_interrupt r_intfo_interrupt	× ○ ○ ○ ○
	r_cg_intc.h	－	－

表 2.12　出力ファイル(12/15)

周辺機能	ファイル名	API 関数名	出力 (*1)
キー割り込み機能	r_cg_intc.c または r_cg_key.c	R_KEY_Create	○
		R_KEY_Start	○
		R_KEY_Stop	○
	r_cg_key_user.c	R_KEY_Create_UserInit	×
		r_key_interrupt	○
	r_cg_opamp.h	－	－
電圧検出回路	r_cg_doc.c	R_LVD_Create	○
		R_LVD_InterruptMode_Start	○
		R_LVD_Start_VDD	○
		R_LVD_Start_VBAT	○
		R_LVD_Start_VRTC	○
		R_LVD_Start_EXLVD	○
		R_LVD_Stop_VDD	○
		R_LVD_Stop_VBAT	○
		R_LVD_Stop_VRTC	○
		R_LVD_Stop_EXLVD	○
		R_LVI_Create	○
		R_LVI_InterruptMode_Start	○
	r_cg_doc_user.c	R_LVD_Create_UserInit	×
		r_lvd_interrupt	○
		r_lvd_vddinterrupt	○
		r_lvd_vbatinterrupt	○
		r_lvd_vrtcinterrupt	○
		r_lvd_exlvdinterrupt	○
		R_LVI_Create_UserInit	×
		r_lvi_interrupt	○
	r_cg_doc.h	－	－
バッテリ・バックアップ機能	r_cg_bup.c	R_BUP_Create	○
		R_BUP_Start	○
		R_BUP_Stop	○
	r_cg_bupt_user.c	R_BUP_Create_UserInit	×
		r_bup_interrupt	○
	r_cg_bupt.h	－	－
発振停止検出回路	r_cg_osdc.c	R_OSDC_Create	○
		R_OSDC_Start	○
		R_OSDC_Stop	○
		R_OSDC_Set_PowerOff	×
		R_OSDC_Reset	×
	r_cg_osdc_user.c	R_OSDC_Create_UserInit	×
		r_osdc_interrupt	○
	r_cg_osdc.h	－	－
SPI インタフェース	r_cg_saic.c	R_SAIC_Create	○
		R_SAIC_Write	○
		R_SAIC_Read	×
		R_SPI_Create	○
		R_SPI_Start	○
		R_SPI_Stop	○
	r_cg_saic_user.c	R_SAIC_Create_UserInit	×
		R_SPI_Create_UserInit	×
	r_cg_saic.h	－	－

表 2.13　出力ファイル(13/15)

周辺機能	ファイル名	API 関数名	出力 (*1)
オペアンプ	r_cg_opamp.c	R_OPAMP_Create	○
		R_OPAMP_Set_ReferenceCircuitOn	○
		R_OPAMP_Set_ReferenceCircuitOff	○
		R_OPAMPn_Start	○
		R_OPAMPn_Stop	○
		R_OPAMPn_Set_PrechargeOn	×
		R_OPAMPn_Set_PrechargeOff	×
	r_cg_opamp_user.c	R_OPAMP_Create_UserInit	×
	r_cg_opamp.h	－	－
データ演算回路	r_cg_doc.c	R_DOC_Create	○
		R_DOC_SetMode	○
		R_DOC_WriteData	○
		R_DOC_GetResult	○
		R_DOC_ClearFlag	○
		R_DOC_Set_PowerOff	×
		R_DOC_Reset	×
	r_cg_doc_user.c	R_DOC_Create_UserInit	×
		r_doc_interrupt	○
	r_cg_doc.h	－	－
32 ビット積和演算回路	r_cg_mac32bit.c	R_MAC32Bit_Create	○
		R_MAC32Bit_Reset	×
		R_MAC32Bit_Set_PowerOff	×
		R_MAC32Bit_MULUnsigned	×
		R_MAC32Bit_MULSigned	×
		R_MAC32Bit_MACUnsigned	×
		R_MAC32Bit_MACSigned	×
	r_cg_mac32bit_user.c	R_MAC32Bit_Create_UserInit	×
		r_mac32bit_interrupt_flow	○
	r_cg_mac32bit.h	－	－
12 ビット A/D コンバータ	r_cg_12adc.c	R_12ADC_Create	○
		R_12ADC_Start	○
		R_12ADC_Stop	○
		R_12ADC_Get_ValueResult	○
		R_12ADC_Set_ADChannel	×
		R_12ADC_TemperatureSensorOutput_On	×
		R_12ADC_TemperatureSensorOutput_Off	×
		R_12ADC_InternalReferenceVoltage_On	×
		R_12ADC_InternalReferenceVoltage_Off	×
		R_12ADC_Set_PowerOff	×
	r_cg_12adc_user.c	R_12ADC_Create_UserInit	×
		r_12adc_interrupt	○
	r_cg_12adc.h	－	－
12 ビット D/A コンバータ	r_cg_12da.c	R_12DA_Create	○
		R_12DAn_Start	○
		R_12DAn_Stop	○
		R_12DA_Set_PowerOff	×
		R_12DAn_Set_ConversionValue	○
	r_cg_12adc_user.c	R_12DA_Create_UserInit	×
	r_cg_12adc.h	－	－

表 2.14　出力ファイル(14/15)

周辺機能	ファイル名	API 関数名	出力 (*1)
オペアンプ&アナログスイッチ	r_cg_ampansw.c	R_AMPANSW_Create	○
		R_OPAMPm_Set_ReferenceCircuitOn	○
		R_OPAMPm_Set_ReferenceCircuitOff	○
		R_OPAMPm_Start	○
		R_OPAMPm_Stop	○
		R_ANSW_ChargePumpm_On	○
		R_ANSW_ChargePumpm_Off	○
	r_cg_ampansw_user.c	R_AMPANSW_Create_UserInit	×
	r_cg_ampansw.h	－	－
ボルテージ・リファレンス	r_cg_vr.c	R_VR_Create	○
		R_VR_Start	○
		R_VR_Stop	○
	r_cg_vr_user.c	R_VR_Create_UserInit	×
	r_cg_vr.h	－	－
サンプリング出力タイマ／ディテクタ	r_cg_smotd.c	R_SMOTD_Create	○
		R_SMOTD_Start	○
		R_SMOTD_Stop	○
		R_SMOTD_Set_PowerOff	×
	r_cg_smotd_user.c	R_SMOTD_Create_UserInit	×
		r_smotd_counterA_interrupt	○
		r_smotd_counterB_interrupt	○
		r_smotd_smpn_interrupt	○
	r_cg_smotd.h	－	－
外部サンプリング	r_cg_exsd.c	R_EXSD_Create	○
		R_EXSD_Start	○
		R_EXSD_Stop	○
		R_EXSD_Set_PowerOff	×
	r_cg_exsd_user.c	R_EXSD_Create_UserInit	×
		r_exsd_interrupt	○
	r_cg_exsd.h	－	－
シリアル・インタフェース UARTMG	r_cg_uartmg.c	R_UARTMGn_Create	○
		R_UARTMGn_Start	○
		R_UARTMGn_Stop	○
		R_UARTMGn_Set_PowerOff	×
		R_UARTMGn_Send	○
		R_UARTMGn_Receive	○
	r_cg_uartmg_user.c	R_UARTMGn_Create_UserInit	×
		r_uartmgn_interrupt_send	○
		r_uartmgn_interrupt_receive	○
		r_uartmgn_interrupt_error	○
		r_uartmgn_callback_sendend	○
		r_uartmgn_callback_receiveend	○
		r_uartmgn_callback_error	○
		r_uartmgn_callback_softwareoverrun	○
	r_cg_uartmg.h	－	－

表 2.15　出力ファイル(15/15)

周辺機能	ファイル名	API 関数名	出力 (*1)
アンプ・ユニット	r_cg_amp.c	R_AMP_Create	○
		R_AMP_Set_PowerOn	○
		R_AMP_Set_PowerOff	○
		R_PGA1_Start	○
		R_PGA1_Stop	○
		R_AMPn_Start	○
		R_AMPn_Stop	○
	r_cg_amp_user.c	R_AMP_Create_UserInit	×
	r_cg_ampr.h	－	－
データフラッシュライブラリ	r_cg_fdl.c	R_FDL_Create	○
		R_FDL_Open	○
		R_FDL_Close	○
		R_FDL_Write	○
		R_FDL_Read	○
		R_FDL_Erase	○
	r_cg_fdl.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。
*2　GNUコンパイラ用のコード生成時
*3　IAR製コンパイラ用のコード生成時
*4　LLVMコンパイラ用のコード生成時

3. API 関数

本章では、コード生成ツールが出力する API 関数について説明します。

3.1 概 要

以下に、コード生成ツールが 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。

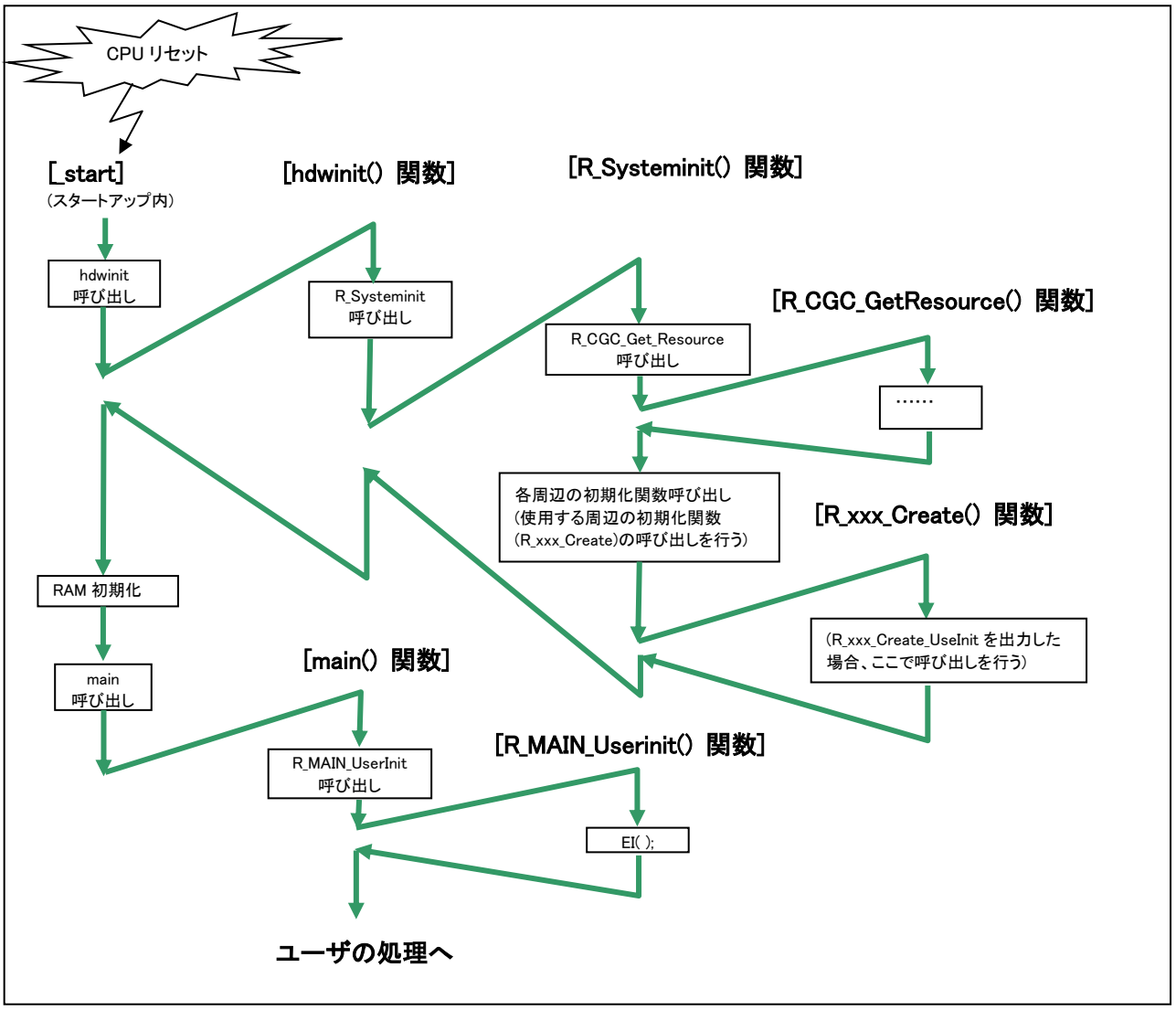
備考 コード生成ツールが生成するコードには、レジスタの反映待ち処理等でfor 文、while 文、do while 文（ループ処理）を使用している関数があります。
無限ループに対するフェイルセーフ処理が必要な場合は、生成されたコードを確認の上、処理を追加してください。

3.2 初期化処理

本節では、main()関数までの初期化フローについて説明します。

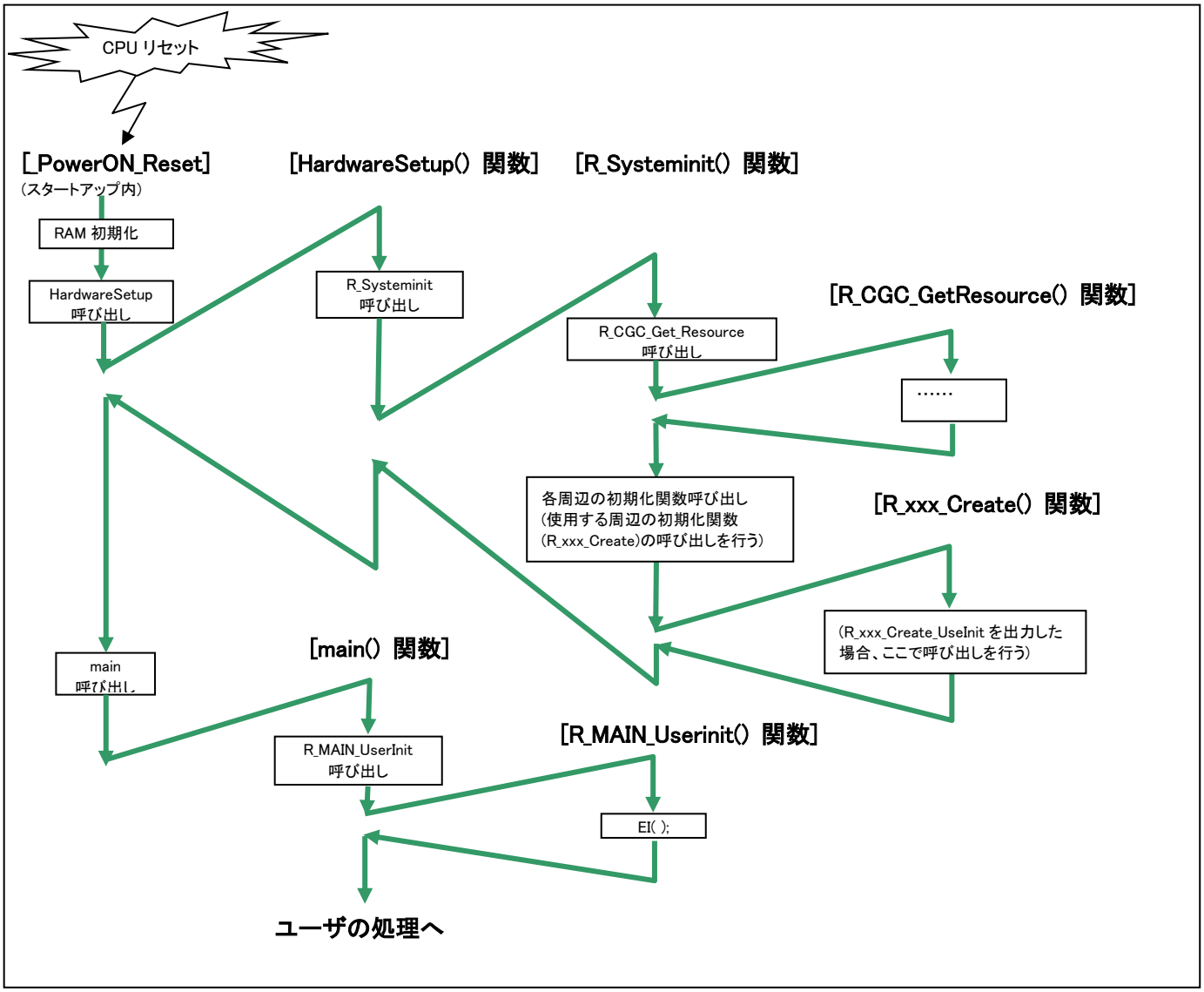
3.2.1 ルネサス製コンパイラ用

図 3.1 初期化フロー(ルネサス製コンパイラ用)



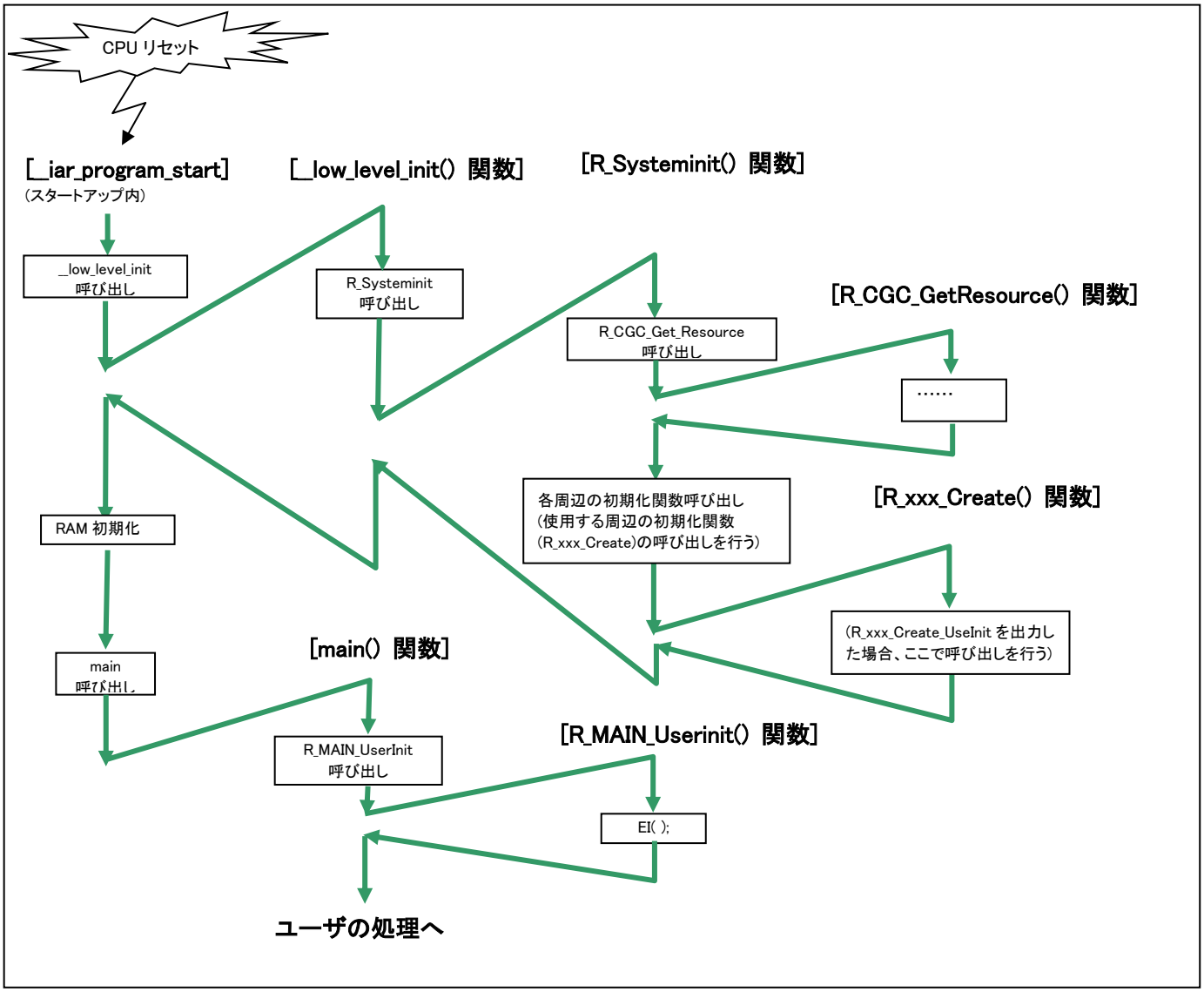
3.2.2 GNU/LLVM コンパイラ用

図 3.2 初期化フロー(GNU/LLVM コンパイラ用)



3.2.3 IAR 製コンパイラ用

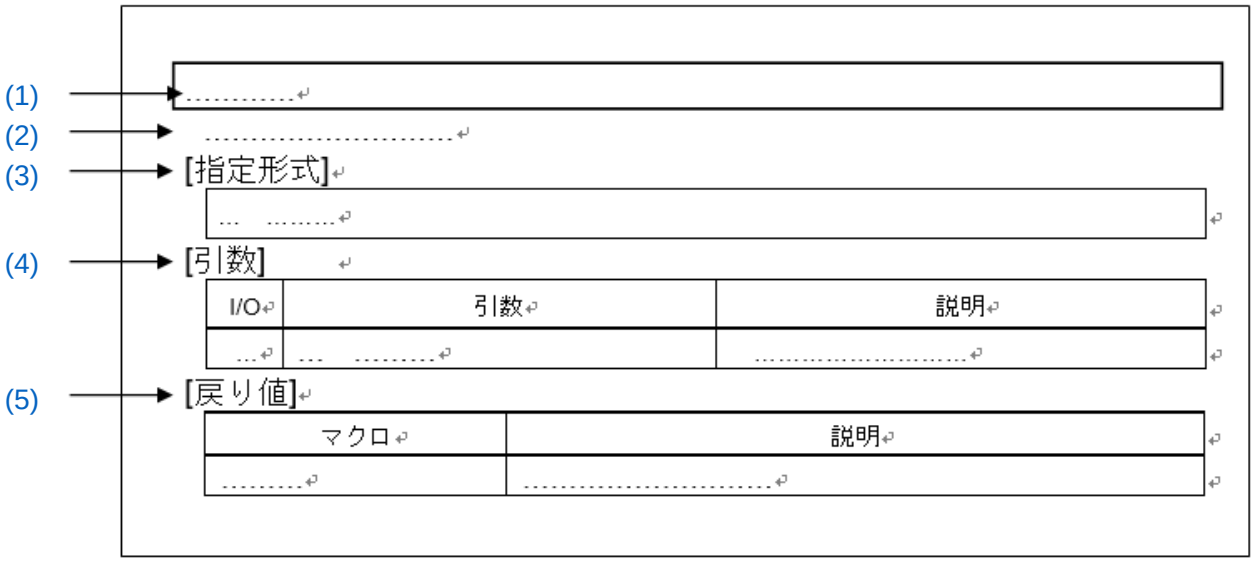
図 3.3 初期化フロー(IAR 製コンパイラ用)



3.3 関数リファレンス

本節では、コード生成ツールが出力する 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 3.4 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

3.3.1 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
hdwinit	各種ハードウェアを制御するうえで必要となる初期化処理を行います。 ルネサス製コンパイラ提供のスタートアップ・ルーチンから自動で呼ばれます。
__low_level_init	各種ハードウェアを制御するうえで必要となる初期化処理を行います。 IAR 製コンパイラ提供のスタートアップ・ルーチンから自動で呼ばれます。
HardwareSetup	各種ハードウェアを制御するうえで必要となる初期化処理を行います。 GNU 製コンパイラ用のスタートアップ・ルーチン(r_reset_program.asm)から自動で呼ばれます。
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_UserserInit	ユーザ独自の初期化処理を行います。

hdwinit

__low_level_init

HardwareSetup

各種ハードウェ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 関数の呼び出しは、スタートアップ・ルーチンから呼び出されます。

[指定形式]

```
void    hdwinit( void );
```

```
int     __low_level_init( void );
```

```
int     HardwareSetup( void );
```

[引数]

なし

[戻り値]

hdwinit は、なし

値	説明
1U	正常終了

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、hdwinit 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 関数の呼び出しは、スタートアップ・ルーチンから行ってください。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_MAIN_UserserInit

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

3.3.2 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（リセット機能，オンチップ・デバッグ機能などを含む）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（リセット機能，オンチップ・デバッグ機能などを含む）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（リセット機能，オンチップ・デバッグ機能などを含む）に関するユーザ独自の初期化処理を行います。
r_cgc_ram_ecc_interrupt	RAM1bit 訂正／2bit エラー検出割り込み INTRAM の発生に伴う処理を行います。
r_cgc_stackpointer_interrupt	スタック・ポインタ・オーバフロー／アンダフロー割り込み INTSPM の発生に伴う処理を行います。
r_cgc_clockmonitor_interrupt	クロック･モニタ割り込み INTCLM の発生に伴う処理を行います。
R_CGC_Get_ResetSource	内部リセットの発生に伴う処理を行います。
R_CGC_Set_ClockMode	CPU クロック／周辺ハードウェア・クロックを変更します。
R_CGC_RAMECC_Start	RAM-ECC 機能を開始します。
R_CGC_RAMECC_Stop	RAM-ECC 機能を終了します。
R_CGC_StackPointer_Start	CPU スタック・ポインタ・モニタ機能を開始します。
R_CGC_StackPointer_Stop	CPU スタック・ポインタ・モニタ機能を終了します。
R_CGC_ClockMonitor_Start	クロック・モニタを開始します。
R_CGC_ClockMonitor_Stop	クロック・モニタを終了します。

R_CGC_Create

クロック発生回路（リセット機能，オンチップ・デバッグ機能などを含む）を制御するうえで必要となる初期化処理を行います。

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

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_CGC_Create_UserInit

クロック発生回路（リセット機能，オンチップ・デバッグ機能などを含む）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_ram_ecc_interrupt

RAM1bit 訂正／2bit エラー検出割り込み INTRAM の発生に伴う処理を行います。

備考　本 API 関数は、RAM1bit 訂正／2bit エラー検出割り込み INTRAM 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]

CA78K0R コンパイラの場合

```
__interrupt static void      r_cgc_ram_ecc_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void      __near r_cgc_ram_ecc_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_cgc_stackpointer_interrupt

スタック・ポインタ・オーバフロー／アンダフロー割り込み INTSPM の発生に伴う処理を行います。

備考　本 API 関数は、スタック・ポインタ・オーバフロー／アンダフロー割り込み INTSPM 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]

CA78K0R コンパイラの場合

```
__interrupt static void      r_cgc_stackpointer_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void      __near r_cgc_stackpointer_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_cgc_clockmonitor_interrupt

クロック・モニタ割り込み INTCLM の発生に伴う処理を行います。

備考 本 API 関数は、クロック・モニタ割り込み INTCLM 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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_cgc_clockmonitor_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_cgc_clockmonitor_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_CGC_Get_ResetSource

内部リセットの発生に伴う処理を行います。

[指定形式]

```
void        R_CGC_Get_ResetSource ( void );
```

[引数]

なし

[戻り値]

なし

R_CGC_Set_ClockMode

CPU クロック／周辺ハードウェア・クロックを変更します。

[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*;	CPU クロック／周辺ハードウェア・クロックの種類 HIOCLK : 高速オンチップ・オシレータ SYSX1CLK : X1 クロック SYSEXTCLK : 外部メイン・システム・クロック SUBXT1CLK : XT1 クロック SUBEXTCLK : 外部サブシステム・クロック

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR1	異常終了
MD_ERROR2	異常終了
MD_ERROR3	異常終了
MD_ERROR4	異常終了
MD_ARGERROR	引数の指定が不正

R_CGC_RAMECC_Start

RAM-ECC 機能を開始します。

[指定形式]

```
void        R_CGC_RAMECC_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_CGC_RAMECC_Stop

RAM-ECC 機能を終了します

[指定形式]

```
void        R_CGC_RAMECC_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_CGC_StackPointer_Start

CPU スタック・ポインタ・モニタ機能を開始します。

[指定形式]

```
void        R_CGC_StackPointer_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_CGC_StackPointer_Stop

CPU スタック・ポインタ・モニタ機能を終了します。

[指定形式]

```
void        R_CGC_StackPointer_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_CGC_ClockMonitor_Start

クロック・モニタを開始します。

[指定形式]

```
void        R_CGC_ClockMonitor_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_CGC_ClockMonitor_Stop

クロック・モニタを終了します

[指定形式]

```
void        R_CGC_ClockMonitor_Stop ( void );
```

[引数]

なし

[戻り値]

なし

使用例

外部入力でクロックを切り替える

[GUI 設定例]

クロック発生回路				使用する
	CGC			使用する
			動作モード設定	高速メイン・モード 4.0(V) ≦ VDD ≦ 5.5(V)
			メイン・システム・クロック(fMAIN)設定	高速オンチップオシレータクロック(fIH)
			fIH 動作	使用する
			fIH 周波数	64(MHz)
			fMX 動作	使用する
			高速システム・クロック設定	X1 発振(fX)
			fMX 周波数	4(MHz)
			発振安定時間	65536 (2^18/fX)(μs)
			fPLL 動作	使用しない
			メイン/PLL 選択クロック(fMP)設定	64 (fMAIN)(MHz)
			fSUB 動作	使用する
			サブシステム・クロック(fSUB)設定	XT1 発振(fXT)
			fSUB 周波数	32.768(kHz)
			XT1 発振回路の発振モード選択	低消費発振
			STOP	HALT モード時のクロック供給設定
			低速内蔵発振クロック(fIL)設定	15(kHz)
			低速オンチップ・オシレータクロック(fSL)設定	32.768 (fSUB)(kHz)
			WDT 専用低速オンチップ・オシレータ・クロック(fWDT)設定	15(kHz)
			RTC 動作クロック	32.768 (fSUB)(kHz)
			タイマ RD 動作クロック	64000 (fIH)(kHz)
			CPU と周辺クロック(fCLK)	32000 (fMP/2)(kHz)

割り込み				使用する
	INTP			使用する
		INTP0		
			有効エッジ	立下りエッジ
			優先順位	低

[API 設定例]

r_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Clear INTP0 interrupt flag and enable interrupt */
    R_INTC0_Start();

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

r_cg_intc_user.c

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

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

static void __near r_intc0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Change clock generator operation mode */
    if (0U == g_cgc_f)
    {
        if (MD_OK == R_CGC_Set_ClockMode(SUBXT1CLK))
        {
            g_cgc_f = 1U;
        }
    }
    else
    {
        if (MD_OK == R_CGC_Set_ClockMode(HIOCLK))
        {
            g_cgc_f = 0U;
        }
    }
    /* End user code. Do not edit comment generated here */
}
```

3.3.3 ポート機能

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

表 3.3 ポート機能用 API 関数

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

R_PORT_Create

ポート機能を制御するうえで必要となる初期化処理を行います。

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

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_PORT_Create_UserInit

ポート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_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

3.3.4 高速オンチップ・オシレータ・クロック周波数補正機能

以下に、コード生成ツールが高速オンチップ・オシレータ・クロック周波数補正機能用として出力する API 関数の一覧を示します。

表 3.4 高速オンチップ・オシレータ・クロック周波数補正機能用 API 関数

API 関数名	機能概要
R_HOFC_Create	高速オンチップ・オシレータ・クロック周波数補正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_HOFC_Create_UserInit	高速オンチップ・オシレータ・クロック周波数補正機能に関するユーザ独自の初期化処理を行います。
r_hofc_interrupt	高速オンチップ・オシレータ・クロック周波数補正機能完了割り込みの発生に伴う処理を行います。
R_HOFC_Start	高速オンチップ・オシレータ・クロック周波数補正機能を開始します。
R_HOFC_Stop	高速オンチップ・オシレータ・クロック周波数補正機能を終了します。

R_HOFC_Create

高速オンチップ・オシレータ・クロック周波数補正機能を制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_HOFC_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_HOFC_Create_UserInit

高速オンチップ・オシレータ・クロック周波数補正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_HOFC_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_HOFC_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_hofc_interrupt

高速オンチップ・オシレータ・クロック周波数補正機能完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、高速オンチップ・オシレータ・クロック周波数補正機能完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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_hofc_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void     _near r_hofc_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_HOFC_Start

高速オンチップ・オシレータ・クロック周波数補正機能を開始します。

[指定形式]

```
void        R_HOFC_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_HOFC_Stop

高速オンチップ・オシレータ・クロック周波数補正機能を終了します。

[指定形式]

```
void        R_HOFC_Stop ( void );
```

[引数]

なし

[戻り値]

なし

3.3.5 タイマ・アレイ・ユニット

以下に、コード生成ツールがタイマ・アレイ・ユニット用として出力する API 関数の一覧を示します。

表 3.5 タイマ・アレイ・ユニット用 API 関数

API 関数名	機能概要
R_TAUm_Create	タイマ・アレイ・ユニッ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_TAUm_Create_UserInit	タイマ・アレイ・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_taum_channeln_interrupt	タイマ割り込み INTTM*mn* の発生に伴う処理を行います。
r_taum_channeln_higher8bits_interrupt	タイマ割り込み INTTM*mn*H の発生に伴う処理を行います。
R_TAUm_Channeln_Start	チャネル *n* のカウントを開始します。
R_TAUm_Channeln_Higher8bits_Start	チャネル *n* のカウント（上位 8 ビット）を開始します。
R_TAUm_Channeln_Lower8bits_Start	チャネル *n* のカウント（下位 8 ビット）を開始します。
R_TAUm_Channeln_Stop	チャネル *n* のカウントを終了します。
R_TAUm_Channeln_Higher8bits_Stop	チャネル *n* のカウント（上位 8 ビット）を終了します。
R_TAUm_Channeln_Lower8bits_Stop	チャネル *n* のカウント（下位 8 ビット）を終了します。
R_TAUm_Reset	タイマ・アレイ・ユニットをリセットします。
R_TAUm_Set_PowerOff	タイマ・アレイ・ユニットに対するクロック供給を停止します。
R_TAUm_Channeln_Get_PulseWidth	TI*mn* 端子に対する入力信号（入力パルス）のパルス間隔、またはハイ／ロウ・レベルの測定幅を取得します。
R_TAUm_Channeln_Set_SoftwareTriggerOn	ワンショット・パルス出力のためのトリガ（ソフトウェア・トリガ）を発生させます。

R_TAU*m*_Create

タイマ・アレイ・ユニットを制御するうえで必要となる初期化処理を行います。

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

[指定形式]

void	R_TAU*m*_Create (void);

備考　*m* は、ユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_TAU*m*_Create_UserInit

タイマ・アレイ・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_TAUm_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_TAU*m*_Create_UserInit (void);

備考　*m* は、ユニット番号を意味します。

[引数]

なし

[戻り値]

なし

r_tau*m*_channel*n*_interrupt

タイマ割り込み INTTM*mn* の発生に伴う処理を行います。

備考　本 API 関数は、タイマ割り込み INTTM*mn* 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_tau*m*_channel*n*_interrupt (void);

備考　*m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

r_tau*m*_channel*n*_higher8bits_interrupt

タイマ割り込み INTTM*mn*H 割り込みの発生に伴う処理を行います。

備考　本 API 関数は、タイマ割り込み INTTM*mn*H 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_taum_channeln_higher8bits_interrupt (void);

備考　*m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_TAU*m*_Channel*n*_Start

チャネル *n* のカウントを開始します。

備考 1. 本 API 関数を呼び出してからカウント処理を開始するまでの時間は、当該機能の種類（インターバル・タイマ，方形波出力，外部イベント・カウンタなど）により異なります。

備考 2, タイマを停止して再開（再度 R_TAU*m*_Channel*n*_Start）させた場合、カウンタ値は TDR レジスタから TCR レジスタへ再ロードされます。そのため、タイマは初期値で設定した値になります。

[指定形式]

void	R_TAU*m*_Channel*n*_Start (void);

備考 *m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_TAU*m*_Channel*n*_Higher8bits_Start

チャネル *n* のカウント（上位 8 ビット）を開始します。

備考　本 API 関数を呼び出しは、タイマ・アレイ・ユニットを 8 ビット・タイマとして使用している場合に限られます。

[指定形式]

void	R_TAU*m*_Channel*n*_Higher8bits_Start (void);

備考　*m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_TAU*m*_Channel*n*_Lower8bits_Start

チャネル *n* のカウント（下位 8 ビット）を開始します。

備考 1. 本 API 関数を呼び出しは、タイマ・アレイ・ユニットを 8 ビット・タイマとして使用している場合に限られます。

備考 2. 本 API 関数を呼び出してからカウント処理を開始するまでの時間は、当該機能の種類（インターバル・タイマ，外部イベント・カウンタ，ディレイ・カウンタなど）により異なります。

[指定形式]

void	R_TAU*m*_Channel*n*_Lower8bits_Start (void);

備考 *m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_TAU*m*_Channel*n*_Stop

チャネル *n* のカウントを終了します。

備考　タイマを停止して再開（再度 R_TAU*m*_Channel*n*_Start）させた場合、カウンタ値は TDR レジスタから TCR レジスタへ再ロードされます。そのため、タイマは初期値で設定した値になります。

[指定形式]

void	R_TAU*m*_Channel*n*_Stop (void);

備考　*m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_TAU*m*_Channel*n*_Higher8bits_Stop

チャネル *n* のカウント（上位 8 ビット）を終了します。

備考　本 API 関数を呼び出しは、タイマ・アレイ・ユニットを 8 ビット・タイマとして使用している場合に限られます。

[指定形式]

void	R_TAU*m*_Channel*n*_Higher8bits_Stop (void);

備考　*m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_TAU*m*_Channel*n*_Lower8bits_Stop

チャネル *n* のカウント（下位 8 ビット）を終了します。

備考　本 API 関数を呼び出しは、タイマ・アレイ・ユニットを 8 ビット・タイマとして使用している場合に限られます。

[指定形式]

void	R_TAU*m*_Channel*n*_Lower8bits_Stop (void);

備考　*m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_TAU*m*_Reset

タイマ・アレイ・ユニットをリセットします。

[指定形式]

void	R_TAU*m*_Reset (void);

備考 *m* は、ユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_TAU*m*_Set_PowerOff

タイマ・アレイ・ユニットに対するクロック供給を停止します。

備考　本 API 関数を呼び出しにより、タイマ・アレイ・ユニットはリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

void	R_TAU*m*_Set_PowerOff (void);

備考　*m* は、ユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_TAU*m*_Channel*n*_Get_PulseWidth

TI*mn* 端子に対する入力信号（入力パルス）のパルス間隔、またはハイ／ロウ・レベルの測定幅を取得します。

[指定形式]

#include	“r_cg_macrodriver.h”
void	R_TAU*m*_Channel*n*_Get_PulseWidth (uint32_t * const width);
備考	*m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

I/O	引数	説明
O	Unit32_t * const *width*;	測定幅（0x0 ～ 0x1FFFF）を格納する領域へのポインタ

[戻り値]

なし

R_TAU*m*_Channeln_Set_SoftwareTriggerOn

ワンショット・パルス出力のためのトリガ（ソフトウェア・トリガ）を発生させます。

[指定形式]

void	R_TAU*m*_Channeln_Set_SoftwareTriggerOn (void);
備考	*m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

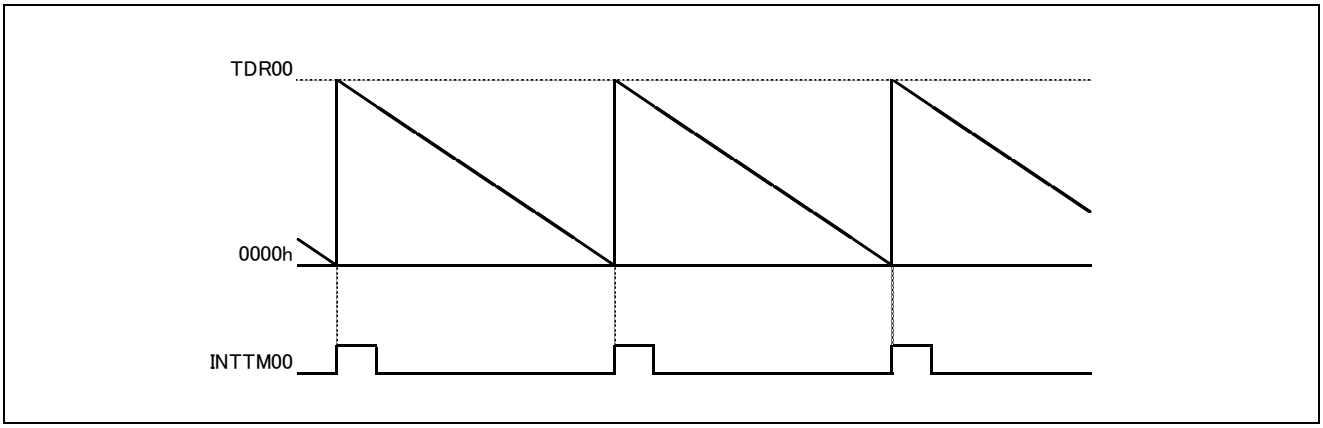
[戻り値]

なし

使用例(インターバル・タイマ)

一定間隔で割り込み関数に入り、その回数をカウントする

[波形例]



[GUI 設定例]

タイマ				使用する
	TAU0			使用する
		Channel0		
			チャネル 0	インターバル・タイマ
			インターバル時間(16 ビット)	100µs (実際の値：100)
			カウント開始時に INTTM00 割り込みを発生する	使用しない
			タイマ・チャネル 0 のカウント完了で割り込み発生(INTTM00)	使用する
			優先順位 (INTTM00)	低

[API 設定例]

r_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start TAU0 channel 0 counter */
    R_TAU0_Channel0_Start();

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

r_cg_timer_user.c

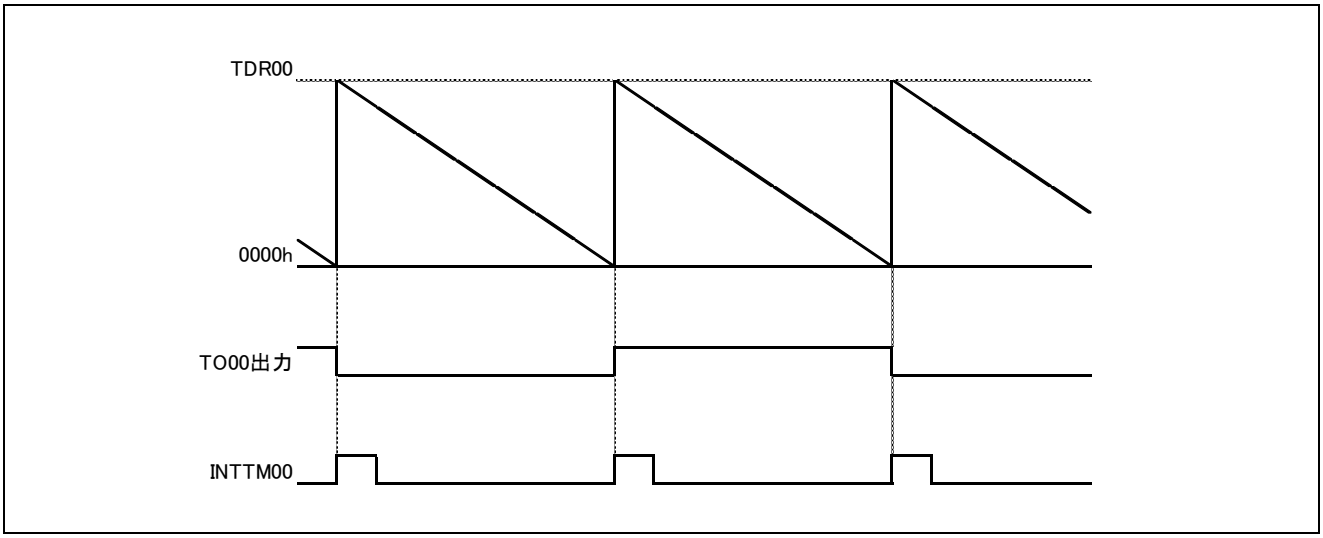
```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_cnt = 0U;
/* End user code. Do not edit comment generated here */

static void __near r_tau0_channel0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Count INTTM00 */
    g_cnt++;
    /* End user code. Do not edit comment generated here */
}
```

使用例(方形波出力)

一定間隔でトグル動作を行い，デューティ 50%の方形波を出力する

[波形例]



[GUI 設定例]

タイマ				使用する
	TAU0			使用する
		Channel0		
			チャネル 0	方形波出力
			方形波幅	100μs (実際の値：100)
			カウント開始時に INTTM00 を発生し，タイマ出力を反転する	使用しない
			初期出力値	0
			タイマ・チャネル 0 のカウント完了で割り込み発生(INTTM00)	使用する
			優先順位 (INTTM00)	低

[API 設定例]

r_main.c

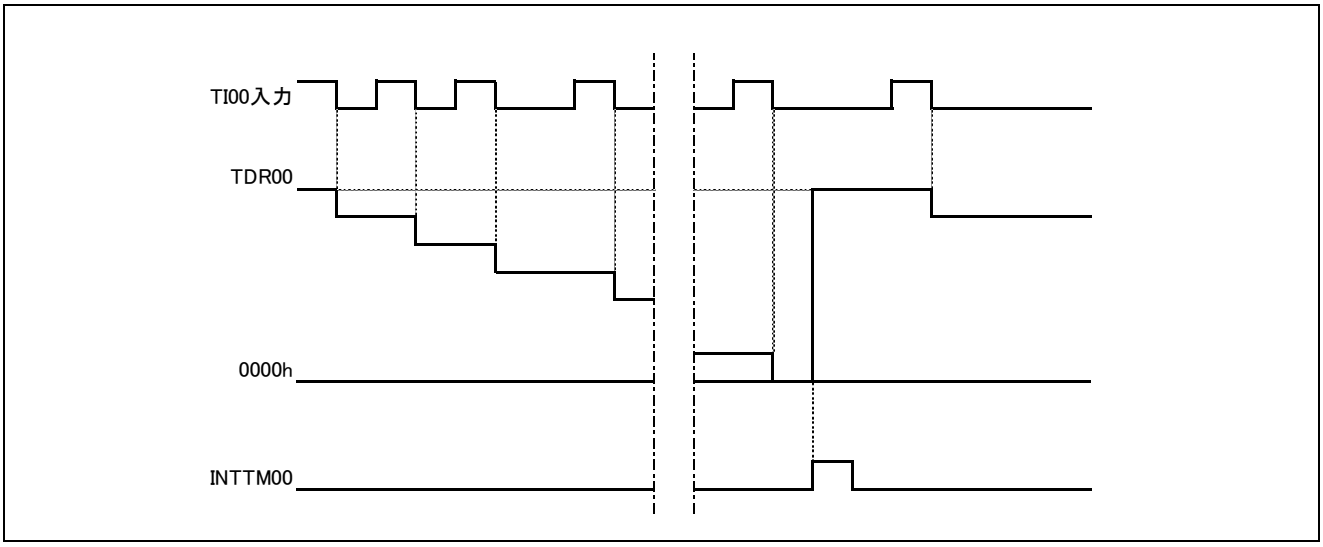
```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start TAU0 channel 0 counter */
    R_TAU0_Channel0_Start();

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

使用例(外部イベント・カウンタ)

立ち下がりエッジ 100 回毎にカウントする

[波形例]



[GUI 設定例]

タイマ				使用する
	TAU0			使用する
		Channel0		
			チャネル 0	外部イベント・カウンタ
			TI00 入力可能な最大周波数	16000000 (Hz)
			TI00 端子入力信号のノイズ・フィルタ使用	使用しない
			外部イベント選択	TI00 立下りエッジ
			カウント値	100
			タイマ・チャネル 0 のカウント完了で割り込み発生(INTTM00)	使用する
			優先順位	低

[API 設定例]

r_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start TAU0 channel 0 counter */
    R_TAU0_Channel0_Start();

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

r_cg_timer_user.c

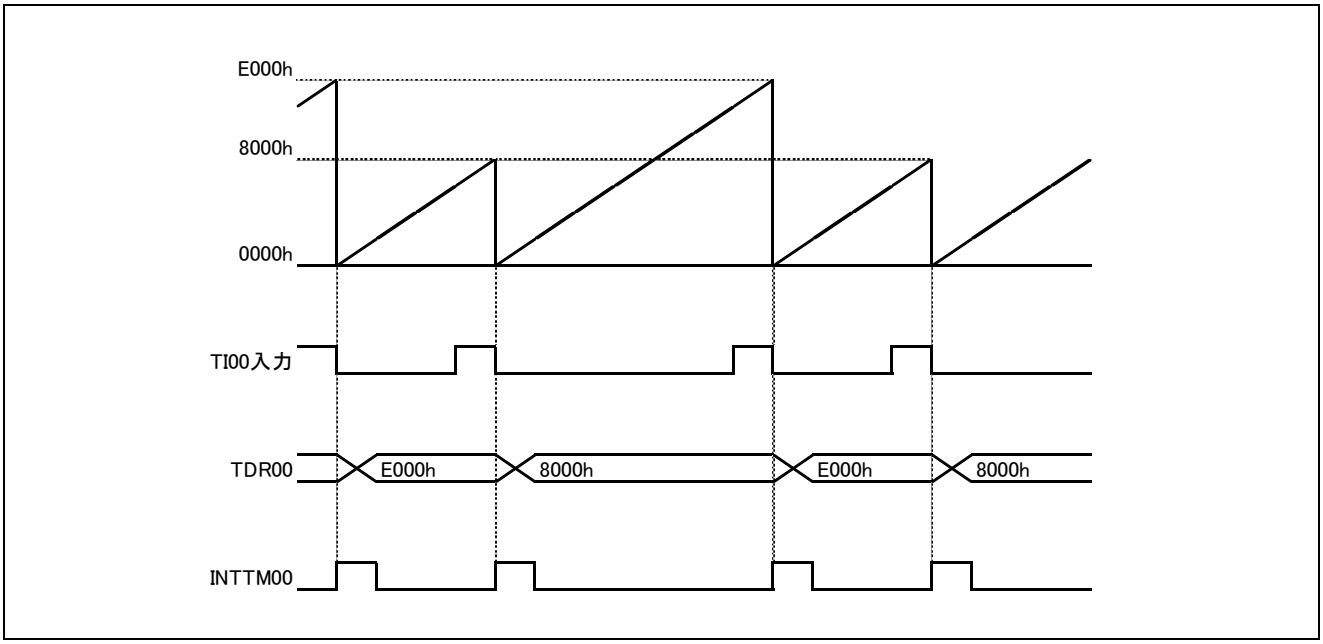
```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_cnt = 0U;
/* End user code. Do not edit comment generated here */

static void __near r_tau0_channel0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Count INTTM00 */
    g_cnt++;
    /* End user code. Do not edit comment generated here */
}
```

使用例(入力パルス間隔測定)

入力端子の立下りエッジの間隔を測定する

[波形例]



[GUI 設定例]

タイマ				使用する
	TAU0			使用する
		Channel 0		
			チャネル 0	入力パルス間隔測定
			入力ソース設定	TI00
			TI00 測定可能なパルス間隔	0.125 (μs) < TI00 < 8.192 (ms)
			TI00 端子入力信号のノイズ・フィルタ使用	使用しない
			カウント開始時に INTTM00 発生する	使用しない
			入力エッジ設定	立下りエッジ
			タイマ・チャネル 0 のキャプチャ完了 (INTTM00)	使用する
			優先順位	低

備考　カウント・クロックの周期は、「TI00 測定可能なパルス間隔」で選択した範囲の最小値の 1/2 となります。この設定では、0.0625usec となります。

[API 設定例]

r_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start TAU0 channel 0 counter */
    R_TAU0_Channel0_Start();

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

r_cg_timer_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint32_t g_width = 0UL;
/* End user code. Do not edit comment generated here */

static void __near r_tau0_channel0_interrupt(void)
{
    if ((TSR00 & _0001_TAU_OVERFLOW_OCCURS) == 1U)      /* overflow occurs */
    {
        g_tau0_ch0_width = (uint32_t)(TDR00 + 1U) + 0x10000U;
    }
    else
    {
        g_tau0_ch0_width = (uint32_t)(TDR00 + 1U);
    }

    /* Start user code. Do not edit comment generated here */
    /* Get TAU0 channel 0 input pulse width. Pulse width(usec) = (Period of count clock(usec) *
g_width) */
    R_TAU0_Channel0_Get_PulseWidth((uint32_t *)&g_width);
    /* End user code. Do not edit comment generated here */
}
```

3.3.6 タイマ RJ

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

表 3.6 タイマ RJ 用 API 関数

API 関数名	機能概要
R_TMR_RJ0_Create	16 ビット・タイマ RJ0 を制御するうえで必要となる初期化処理を行います。
R_TMR_RJ0_Create_UserInit	16 ビット・タイマ RJ0 に関するユーザ独自の初期化処理を行います。
r_tmr_rj0_interrupt	タイマ割り込みの発生に伴う処理を行います。
R_TMR_RJ0_Start	16 ビット・タイマ RJ0 のカウント処理を開始します。
R_TMR_RJ0_Stop	16 ビット・タイマ RJ0 のカウント処理を終了します。
R_TMR_RJ0_Set_PowerOff	16 ビット・タイマ RJ0 に対するクロック供給を停止します。
R_TMR_RJ0_Get_PulseWidth	16 ビット・タイマ RJ0 のパルス幅を読み出します。
R_TMRJ0_Create	16 ビット・タイマ RJ0 を制御するうえで必要となる初期化処理を行います。
R_TMRJ0_Create_UserInit	16 ビット・タイマ RJ0 に関するユーザ独自の初期化処理を行います。
r_tmrj0_interrupt	タイマ割り込みの発生に伴う処理を行います。
R_TMRJ0_Start	16 ビット・タイマ RJ0 のカウント処理を開始します。
R_TMRJ0_Stop	16 ビット・タイマ RJ0 のカウント処理を終了します。
R_TMRJ0_Set_PowerOff	16 ビット・タイマ RJ0 に対するクロック供給を停止します。
R_TMRJ0_Get_PulseWidth	16 ビット・タイマ RJ0 のパルス幅を読み出します。

R_TMR_RJ0_Create

16 ビット・タイマ RJ0 を制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_TMR_RJ0_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_TMR_RJ0_Create_UserInit

16 ビット・タイマ RJ0 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_RJ0_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_RJ0_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_tmr_rj0_interrupt

タイマ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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_tmr_rj0_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_tmr_rj0_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_TMR_RJ0_Start

16 ビット・タイマ RJ0 のカウント処理を開始します。

[指定形式]

```
void    R_TMR_RJ0_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_TMR_RJ0_Stop

16 ビット・タイマ RJ0 のカウント処理を終了します。

[指定形式]

```
void    R_TMR_RJ0_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_TMR_RJ0_Set_PowerOff

16 ビット・タイマ RJ0 に対するクロック供給を停止します。

備考　本 API 関数を呼び出しにより、16 ビット・タイマ RJ0 はリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

```
void    R_TMR_RJ0_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

R_TMR_RJ0_Get_PulseWidth

16 ビット・タイマ RJ0 のパルス幅を読み出します。

備考 1. 本 API 関数を呼び出しは、16 ビット・タイマ RJ0 をパルス幅測定モード／パルス周期測定モードで使用している場合に限られます。

備考 2. パルス幅測定中にオーバフロー（2 回以上）が発生した場合、正常なパルス幅を読み出すことはできません。

備考 3. パルス周期測定モードで 1 回目の割り込みで得られるデータは、周期幅ではないので無効です。

[指定形式]

```
#include   “r_cg_macrodriver.h”
void       R_TMR_RJ0_Get_PulseWidth ( uint32_t * const active_width );
```

[引数]

I/O	引数	説明
O	unit32_t * const *active_width*;	TRJ0IO 端子から読み出したアクティブレベル幅を格納する領域へのポインタ

[戻り値]

なし

R_TMRJ0_Create

16 ビット・タイマ RJ0 を制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_TMRJ0_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_TMRJ0_Create_UserInit

16 ビット・タイマ RJ0 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_TMRJ0_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_TMRJ0_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_tmrj0_interrupt

タイマ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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_tmrj0_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_tmrj0_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_TMRJ0_Start

16 ビット・タイマ RJ0 のカウント処理を開始します。

[指定形式]

```
void    R_TMRJ0_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_TMRJ0_Stop

16 ビット・タイマ RJ0 のカウント処理を終了します。

[指定形式]

```
void    R_TMRJ0_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_TMRJ0_Set_PowerOff

16 ビット・タイマ RJ0 に対するクロック供給を停止します。

備考　本 API 関数を呼び出しにより、16 ビット・タイマ RJ0 はリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

```
void    R_TMRJ0_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

R_TMRJ0_Get_PulseWidth

16 ビット・タイマ RJ0 のパルス幅を読み出します。

備考 1. 本 API 関数を呼び出しは、16 ビット・タイマ RJ0 をパルス幅測定モード／パルス周期測定モードで使用している場合に限られます。

備考 2. パルス幅測定中にオーバフロー（2 回以上）が発生した場合、正常なパルス幅を読み出すことはできません。

備考 3. パルス周期測定モードで 1 回目の割り込みで得られるデータは、周期幅ではないので無効です。

[指定形式]

```
#include   “r_cg_macrodriver.h”
void       R_TMRJ0_Get_PulseWidth ( uint32_t * const active_width );
```

[引数]

I/O	引数	説明
O	unit32_t * const *active_width*;	TRJ0IO 端子から読み出したアクティブレベル幅を格納する領域へのポインタ

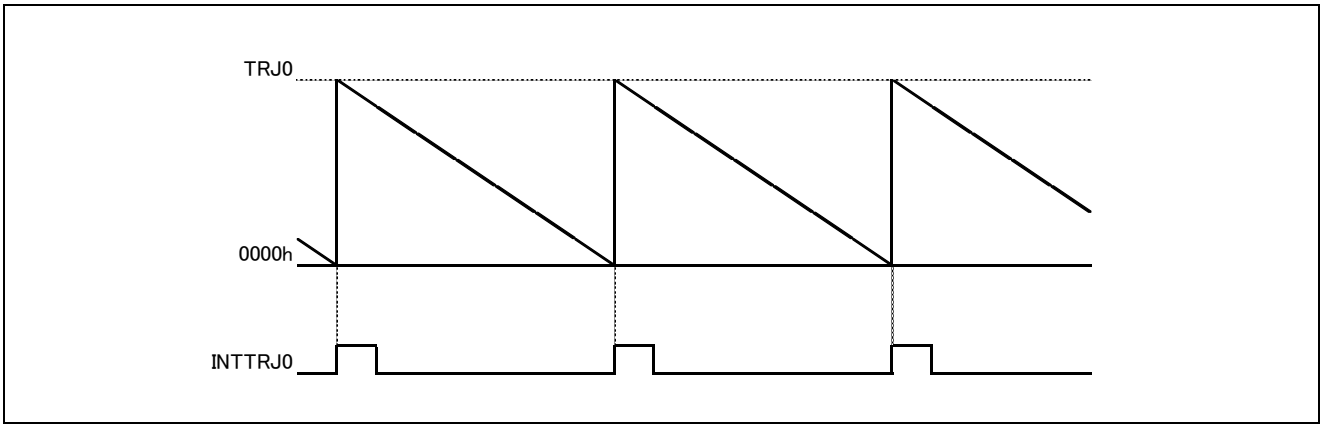
[戻り値]

なし

使用例(タイマモード)

一定間隔で割り込み関数に入り、その回数をカウントする

[波形例]



[GUI 設定例]

タイマ				使用する
	TMRJ0			使用する
			機能	タイマモード
			カウントソース設定	自動
			タイマ値	100μs (実際の値：100)
			カウンタがアンダフローしたとき割り込み発生(INTTRJ0)	使用する
			優先順位	低

[API 設定例]

r_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start TMRJ0 counter */
    R_TMR_RJ0_Start();

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

r_cg_timer_user.c

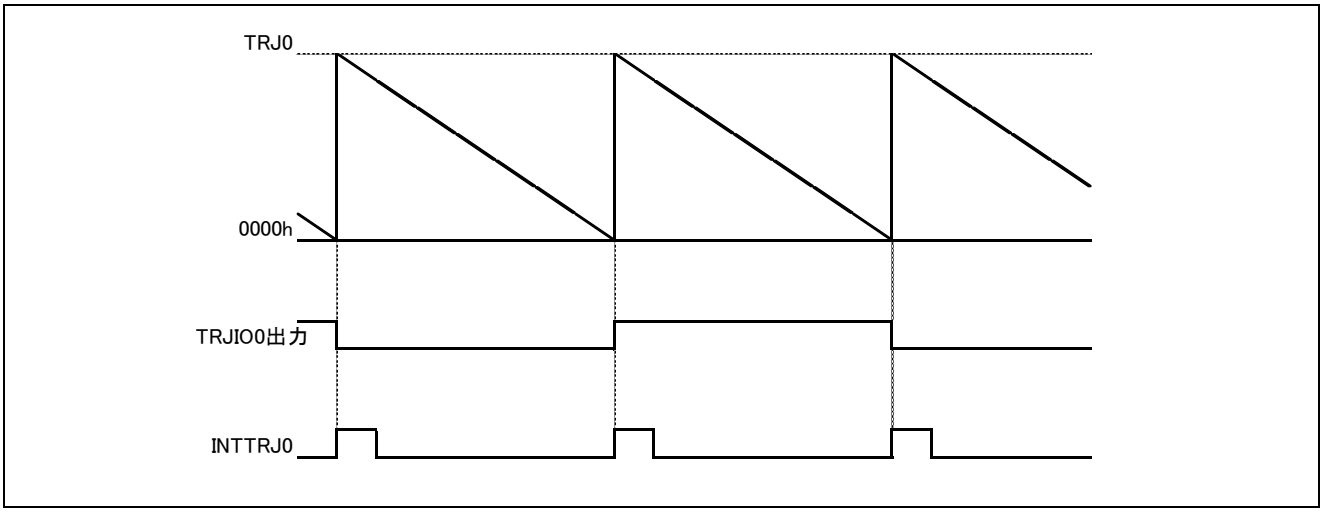
```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_cnt = 0U;
/* End user code. Do not edit comment generated here */

static void __near r_tmr_rj0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Count INTTRJ0 */
    g_cnt++;
    /* End user code. Do not edit comment generated here */
}
```

使用例(パルス出力モード)

一定間隔でトグル動作を行い，デューティ 50%の方形波を出力する

[波形例]



[GUI 設定例]

タイマ				使用する
	TMRJ0			使用する
			機能	パルス出力モード
			カウントソース設定	自動
			タイマ値	100μs (実際の値：100)
			出力(TRIJO0)	“H”から開始
			出力許可(TRJO0)	使用しない
			カウンタがアンダフローしたとき割り込み発生(INTTRJ0)	使用する
			優先順位	低

[API 設定例]

r_main.c

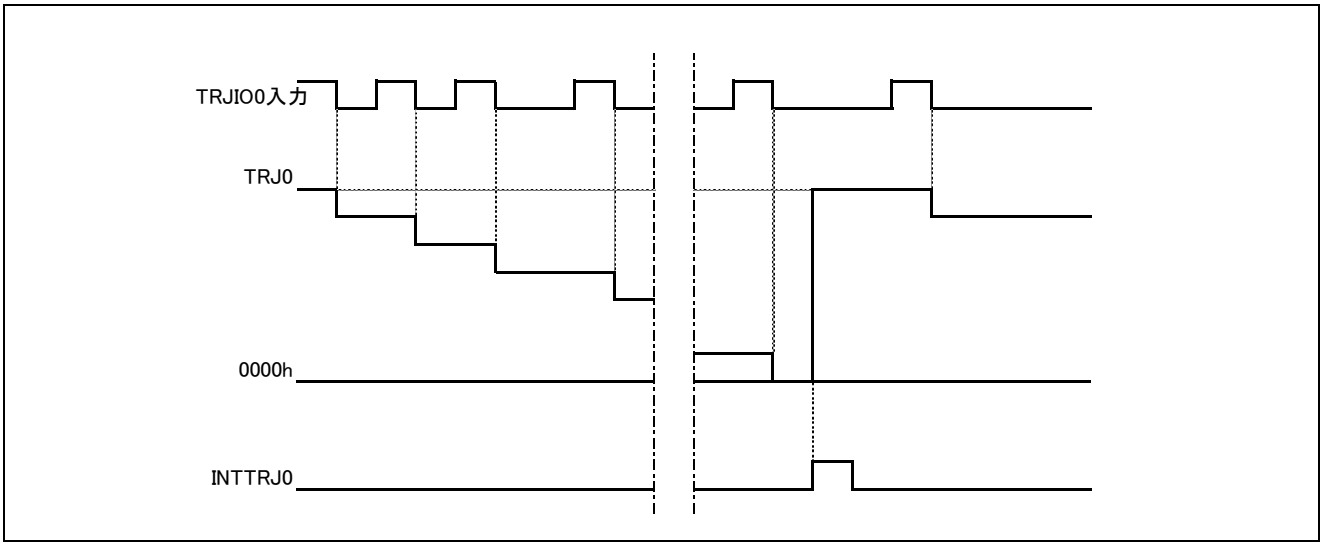
```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start TMRJ0 counter */
    R_TMR_RJ0_Start();

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

使用例(イベントカウンタモード)

立ち下がりエッジ 100 回毎にカウントする

[波形例]



[GUI 設定例]

タイマ				使用する
	TMRJ0			使用する
			機能	イベントカウンタモード
			カウント値	100
			TRJIO0 入力フィルタ使用	使用しない
			TRJIO0 イベント入力	常に有効
			TRJIO0 エッジ極性選択設定	片エッジ
			TRJIO 極性切り替え設定	TRJIO0 入力の立ち下がりエッジでカウント、“H”から TRJO 出力開始
			出力許可(TRJO0)	使用しない
			カウンタがアンダフローしたとき割り込み発生(INTTRJ0)	使用する
			優先順位	低

[API 設定例]

r_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start TMRJ0 counter */
    R_TMR_RJ0_Start();

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

r_cg_timer_user.c

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

static void __near r_tmr_rj0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Count INTTRJ0 */
    g_cnt++;
    /* End user code. Do not edit comment generated here */
}
```

3.3.7 タイマ RD

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

表 3.7 タイマ RD 用 API 関数

API 関数名	機能概要
R_TMR_RDn_Create	16 ビット・タイマ RD*n* を制御するうえで必要となる初期化処理を行います。
R_TMR_RDn_Create_UserInit	16 ビット・タイマ RD*n* に関するユーザ独自の初期化処理を行います。
r_tmr_rdn_interrupt	タイマ割り込みの発生に伴う処理を行います。
R_TMR_RDn_Start	16 ビット・タイマ RD*n* のカウント処理を開始します。
R_TMR_RDn_Stop	16 ビット・タイマ RD*n* のカウント処理を終了します。
R_TMR_RDn_Set_PowerOff	16 ビット・タイマ RD*n* に対するクロック供給を停止します。
R_TMR_RDn_ForcedOutput_Start	16 ビット・タイマ RD*n* のパルス出力強制遮断処理を開始します。
R_TMR_RDn_ForcedOutput_Stop	16 ビット・タイマ RD*n* のパルス出力強制遮断処理を終了します。
R_TMR_RDn_Get_PulseWidth	16 ビット・タイマ RD*n* のパルス幅を読み出します。
R_TMRDn_Create	16 ビット・タイマ RD*n* を制御するうえで必要となる初期化処理を行います。
R_TMRDn_Create_UserInit	16 ビット・タイマ RD*n* に関するユーザ独自の初期化処理を行います。
r_tmrdn_interrupt	タイマ割り込みの発生に伴う処理を行います。
R_TMRDn_Start	16 ビット・タイマ RD*n* のカウント処理を開始します。
R_TMRDn_Stop	16 ビット・タイマ RD*n* のカウント処理を終了します。
R_TMRDn_Set_PowerOff	16 ビット・タイマ RD*n* に対するクロック供給を停止します。
R_TMRDn_ForcedOutput_Start	16 ビット・タイマ RD*n* のパルス出力強制遮断処理を開始します。
R_TMRDn_ForcedOutput_Stop	16 ビット・タイマ RD*n* のパルス出力強制遮断処理を終了します。
R_TMRDn_Get_PulseWidth	16 ビット・タイマ RJ0 のパルス幅を読み出します。
R_TMRD_Set_PowerOff	16 ビット・タイマ RD に対するクロック供給を停止します。
R_TMRD_PWMOP_ForcedOutput_Stop	16 ビット・タイマ RD の PWM 出力強制遮断処理を終了します。
R_TMRD_PWMOP_Set_PowerOff	16 ビット・タイマ RD の PWM・オプション・ユニットに対するクロック供給を停止します。

R_TMR_RD*n*_Create

16 ビット・タイマ RD*n* を制御するうえで必要となる初期化処理を行います。

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

[指定形式]

void	R_TMR_RD*n*_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_TMR_RD*n*_Create_UserInit

16 ビット・タイマ RD*n* 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_RDn_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_RD*n*_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_tmr_rd*n*_interrupt

タイマ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]

CA78K0R コンパイラの場合

```
__interrupt static void    r_tmr_rdn_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void    __near r_tmr_rdn_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_TMR_RD*n*_Start

16 ビット・タイマ RD*n* のカウント処理を開始します。

[指定形式]

```
void    R_TMR_RDn_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_RD*n*_Stop

16 ビット・タイマ RD*n* のカウント処理を終了します。

[指定形式]

```
void    R_TMR_RDn_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_RD*n*_Set_PowerOff

16 ビット・タイマ RD*n* に対するクロック供給を停止します。

備考　本 API 関数を呼び出しにより、16 ビット・タイマ RD*n* はリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

```
void    R_TMR_RDn_Set_PowerOff ( 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_RD*n*_ForcedOutput_Start

16 ビット・タイマ RD*n* のパルス出力強制遮断処理を開始します。

[指定形式]

```
void    R_TMR_RDn_ForcedOurput_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_RD*n*_ForcedOutput_Stop

16 ビット・タイマ RD*n* のパルス出力強制遮断機能を終了します。

備考　本 API 関数の呼び出しは、16 ビット・タイマ RD*n* がカウント停止状態（タイマ RD スタート・レジスタ（TRDSTR）の TSTART ビットが 0）の場合に限られます。

[指定形式]

```
void    R_TMR_RDn_ForcedOutput_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_RD*n*_Get_PulseWidth

16 ビット・タイマ RD*n* のパルス幅を読み出します。

備考 1. 本 API 関数を呼び出しは、16 ビット・タイマ RD*n* をインプット・キャプチャ機能で使用している場合に限られます。

備考 2. パルス幅測定中にオーバフロー（2 回以上）が発生した場合、正常なパルス幅を読み出すことはできません。

[指定形式]

```
#include   “r_cg_macrodriver.h”
#include   “r_cg_timer.h”
void       R_TMR_RDn_Get_PulseWidth ( uint32_t * const active_width,
 uint32_t * const inactive_width, timer_channel_t channel );
```

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	unit32_t * const *active_width*;	読み出したアクティブ・レベル幅を格納する領域へのポインタ
O	uint32_t * const *inactive_width*;	読み出したインアクティブ・レベル幅を格納する領域へのポインタ
I	timer_channel_t *channel*;	読み出し対象端子 TMCHANNELA : TRDIOA*n* 端子 TMCHANNELB : TRDIOB*n* 端子 TMCHANNELC : TRDIOC*n* 端子 TMCHANNELD : TRDIOD*n* 端子

[戻り値]

なし

R_TMRD*n*_Create

16 ビット・タイマ RD*n* を制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_TMRDn_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_TMRD*n*_Create_UserInit

16 ビット・タイマ RD*n* 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_TMRDn_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_TMRDn_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_tmrd*n*_interrupt

タイマ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]

CA78K0R コンパイラの場合

```
__interrupt static void    r_tmrdn_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void    __near r_tmrdn_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_TMRD*n*_Start

16 ビット・タイマ RD*n* のカウント処理を開始します。

[指定形式]

```
void    R_TMRDn_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_TMRD*n*_Stop

16 ビット・タイマ RD*n* のカウント処理を終了します。

[指定形式]

```
void    R_TMRDn_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_TMRD*n*_Set_PowerOff

16 ビット・タイマ RD*n* に対するクロック供給を停止します。

備考　本 API 関数を呼び出しにより、16 ビット・タイマ RD*n* はリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

```
void    R_TMRDn_Set_PowerOff ( 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_TMRD*n*_ForcedOutput_Start

16 ビット・タイマ RD*n* のパルス出力強制遮断処理を開始します。

[指定形式]

```
void    R_TMRDn_ForcedOurput_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_TMRD*n*_ForcedOutput_Stop

16 ビット・タイマ RD*n* のパルス出力強制遮断機能を終了します。

備考　本 API 関数の呼び出しは、16 ビット・タイマ RD*n* がカウント停止状態（タイマ RD スタート・レジスタ（TRDSTR）の TSTART ビットが 0）の場合に限られます。

[指定形式]

```
void    R_TMRDn_ForcedOutput_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_TMRD*n*_Get_PulseWidth

16 ビット・タイマ RD*n* のパルス幅を読み出します。

備考 1. 本 API 関数を呼び出しは、16 ビット・タイマ RD*n* をインプット・キャプチャ機能で使用している場合に限られます。

備考 2. パルス幅測定中にオーバフロー（2 回以上）が発生した場合、正常なパルス幅を読み出すことはできません。

[指定形式]

```
#include   “r_cg_macrodriver.h”
#include   “r_cg_timer.h”
void       R_TMRDn_Get_PulseWidth ( uint32_t * const active_width,
 uint32_t * const inactive_width, timer_channel_t channel );
```

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	unit32_t * const *active_width*;	読み出したアクティブ・レベル幅を格納する領域へのポインタ
O	uint32_t * const *inactive_width*;	読み出したインアクティブ・レベル幅を格納する領域へのポインタ
I	timer_channel_t *channel*;	読み出し対象端子 TMCHANNELA : TRDIOA*n* 端子 TMCHANNELB : TRDIOB*n* 端子 TMCHANNELC : TRDIOC*n* 端子 TMCHANNELD : TRDIOD*n* 端子

[戻り値]

なし

R_TMRD_Set_PowerOff

16 ビット・タイマ RD に対するクロック供給を停止します。

備考　本 API 関数を呼び出しにより、16 ビット・タイマ RD はリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

```
void    R_TMRD_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

R_TMRD_PWMOP_ForcedOutput_Stop

16 ビット・タイマ RD の PWM 出力強制遮断をソフトウェア解除します。

[指定形式]

```
void    R_TMRD_PWMOP_ForcedOutput_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_TMRD_PWMOP_Set_PowerOff

16 ビット・タイマ RD の PWM オプションユニットに対するクロック供給を停止します。

[指定形式]

```
void    R_TMRD_PWMOP_Set_PowerOff ( void );
```

[引数]

なし

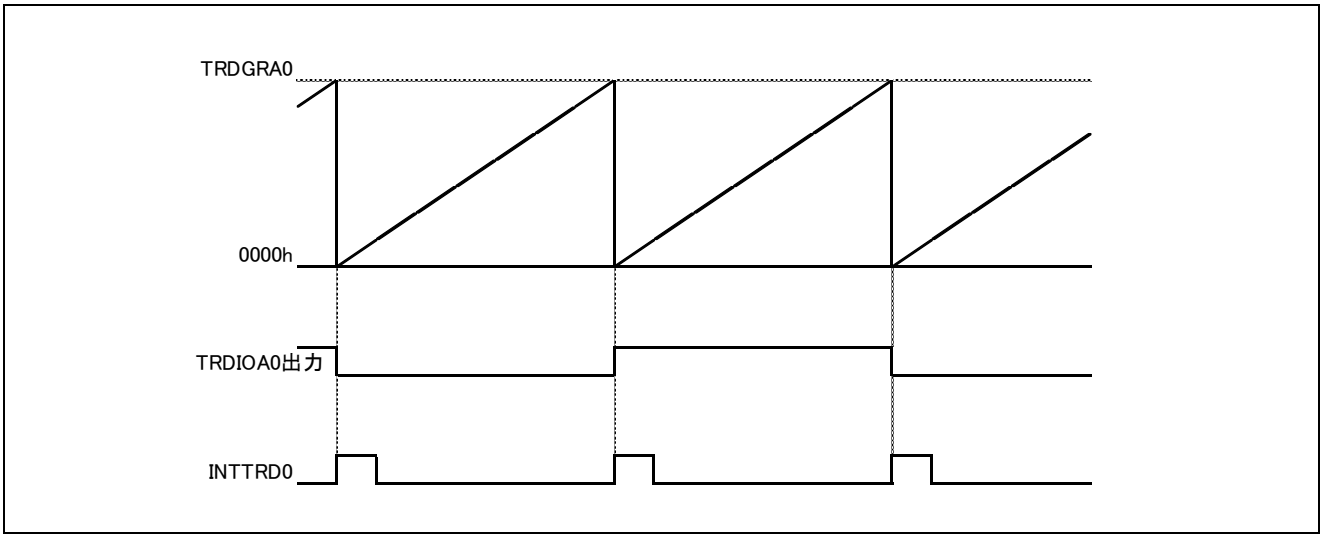
[戻り値]

なし

使用例(アウトプットコンペア機能)

一定間隔でトグル動作を行い，デューティ 50%の方形波を出力する

[波形例]



[GUI 設定例]

タイマ				使用する
	TMRD0			使用する
			アウトプットコンペア機能	使用する
			カウントソース設定	内部クロック
			内部クロック設定	fIH
			カウント動作	TRDGRA0 コンペアー致後もカウント継続
			カウンタクリア	TRDGRA0 コンペアー致でクリア
			レジスタ機能設定 (TRDGRC0)	ジェネラルレジスタ
			レジスタ機能設定 (TRDGRD0)	ジェネラルレジスタ
			コンペア値設定 TRDGRA0	100(μs)(実際の値：100)
			出力設定 TRDIOA0 端子	初期出力 “L” コンペアー致 トグル出力
			TRDGRA0 コンペアー致割り込み許可	使用する

[API 設定例]

r_main.c

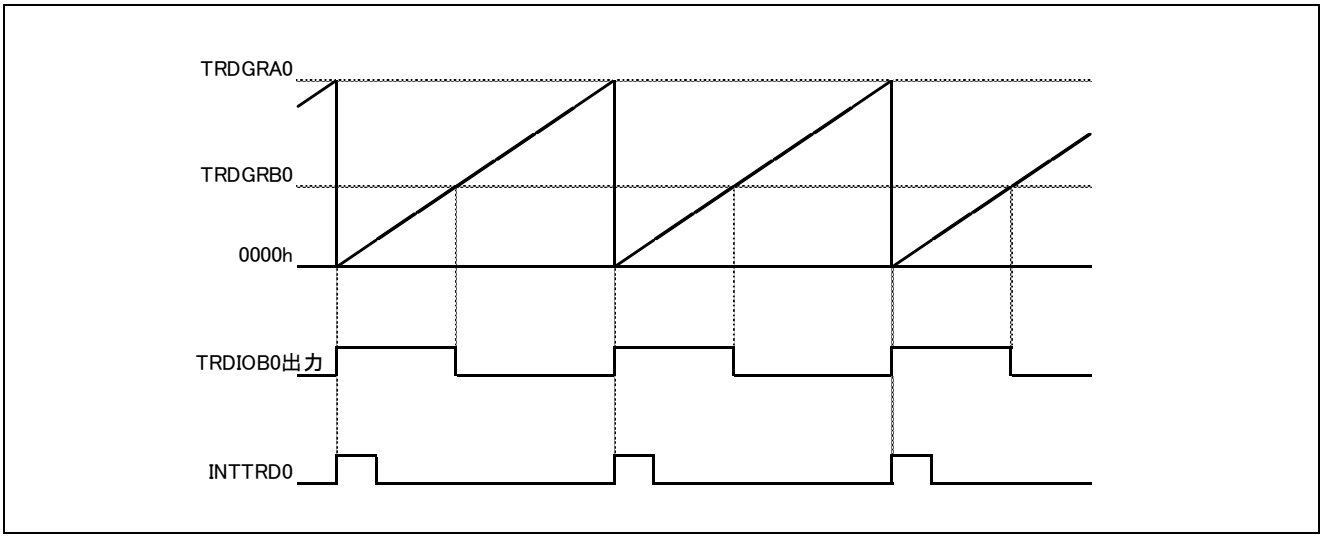
```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start TMRD0 counter */
    R_TMR_RD0_Start();

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

使用例(PWM モード(最大 3 本の PWM 出力))

指定した周期、デューティの PWM 出力を行う

[波形例]



[GUI 設定例]

タイマ				使用する
	TMRD0			使用する
			PWM モード (最大 3 本の PWM 出力)	使用する
			カウントソース設定	内部クロック
			内部クロック設定	fIH
			カウント動作	TRDGRA0 コンペアー致後もカウント継続
			レジスタ機能設定 (TRDGRC0)	ジェネラルレジスタ
			レジスタ機能設定 (TRDGRD0)	ジェネラルレジスタ
			周期	100 (μs) (実際の値：100)
			デューティ (TRDGRB0)	50(%)(実際の値：50)
			出力遅延時間 (TRDGRB0)	遅延なし
			初期出力 (TRDIOB0 端子)	非アクティブレベル
			出力レベル (TRDIOB0 端子)	“L”アクティブ
			ELC イベント入力による強制遮断許可	使用しない
			INTP0 L レベル入力による強制遮断許可	使用しない
			TRDIOB0 端子出力	強制遮断禁止
			TRDIOC0 端子出力	強制遮断禁止
			TRDIOD0 端子出力	強制遮断禁止

			TRDGRA0 コンペアー致割り込み許可	使用する
			TRDGRB0 コンペアー致割り込み許可	使用しない
			TRDGRC0 コンペアー致割り込み許可	使用しない
			TRDGRD0 コンペアー致割り込み許可	使用しない
			TRD0 オーバフロー割り込み許可	使用しない
			優先順位	低

[API 設定例]

r_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start TMRD0 counter */
    R_TMR_RD0_Start();

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

3.3.8 タイマ RG

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

表 3.8 タイマ RG 用 API 関数

API 関数名	機能概要
R_TMR_RG0_Create	16 ビット・タイマ RG0 を制御するうえで必要となる初期化処理を行います。
R_TMR_RG0_Create_UserInit	16 ビット・タイマ RG0 に関するユーザ独自の初期化処理を行います。
r_tmr_rg0_interrupt	タイマ割り込みの発生に伴う処理を行います。
R_TMR_RG0_Start	16 ビット・タイマ RG0 のカウント処理を開始します。
R_TMR_RG0_Stop	16 ビット・タイマ RG0 のカウント処理を終了します。
R_TMR_RG0_Set_PowerOff	16 ビット・タイマ RG0 に対するクロック供給を停止します。
R_TMR_RG0_Get_PulseWidth	16 ビット・タイマ RG0 のパルス幅を読み出します。

R_TMR_RG0_Create

16 ビット・タイマ RG0 を制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_TMR_RG0_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_TMR_RG0_Create_UserInit

16 ビット・タイマ RG0 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_RG0_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_RG0_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_tmr_rg0_interrupt

タイマ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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_tmr_rg0_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_tmr_rg0_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_TMR_RG0_Start

16 ビット・タイマ RG0 のカウント処理を開始します。

[指定形式]

```
void    R_TMR_RG0_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_TMR_RG0_Stop

16 ビット・タイマ RG0 のカウント処理を終了します。

[指定形式]

```
void    R_TMR_RG0_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_TMR_RG0_Set_PowerOff

16 ビット・タイマ RG0 に対するクロック供給を停止します。

備考　本 API 関数を呼び出しにより、16 ビット・タイマ RG0 はリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

```
void    R_TMR_RG0_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

R_TMR_RG0_Get_PulseWidth

16 ビット・タイマ RG0 のパルス幅を読み出します。

備考 1. 本 API 関数を呼び出しは、16 ビット・タイマ RG0 をインプット・キャプチャ機能で使用している場合に限られます。

備考 2. パルス幅測定中にオーバフロー（2 回以上）が発生した場合、正常なパルス幅を読み出すことはできません。

[指定形式]

```
#include   “r_cg_macrodriver.h”
#include   “r_cg_timer.h”
void       R_TMR_RJ0_Get_PulseWidth ( uint32_t * const active_width,
 uint32_t * const inactive_width, timer_channel_t channel );
```

[引数]

I/O	引数	説明
O	unit32_t * const *active_width*;	TRGIO0 端子から読み出したアクティブレベル幅を格納する領域へのポインタ
O	uint32_t * const *inactive_width*;	TRGIO0 端子から読み出したインアクティブ・レベル幅を格納する領域へのポインタ
I	timer_channel_t *channel*;	読み出し対象端子 TMCHANNELA : TRGIOA0 端子 TMCHANNELB : TRGIOB0 端子

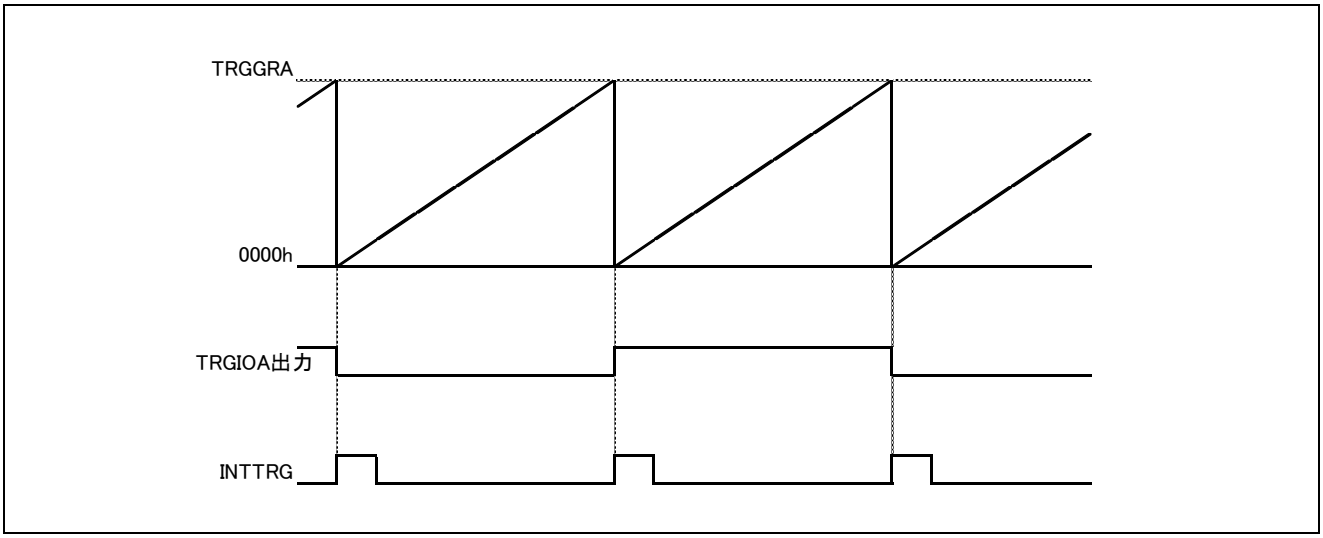
[戻り値]

なし

使用例(アウトプットコンペアモード)

一定間隔でトグル動作を行い，デューティ 50%の方形波を出力する

[波形例]



[GUI 設定例]

タイマ RG				使用する
	TMRG			使用する
			機能	アウトプットコンペア機能
			カウントソース設定	内部クロック
			内部クロック設定	自動
			TRG カウンタ設定 (カウンタクリア)	TRGGRA コンペアー致でクリア
			レジスタ機能設定 (TRGGRC)	ジェネラルレジスタ
			レジスタ機能設定 (TRGGRD)	ジェネラルレジスタ
			コンペア値設定 (TRGGRA)	100μs (実際の値：100)
			出力設定 (TRGIOA 端子)	トグル出力
			TRGGRA コンペアー致割り込み許可	使用する
			TRG オーバフロー割り込み許可	使用しない
			INTTRG 優先順位	レベル 3(低優先順位)

[API 設定例]

r_cg_main.c

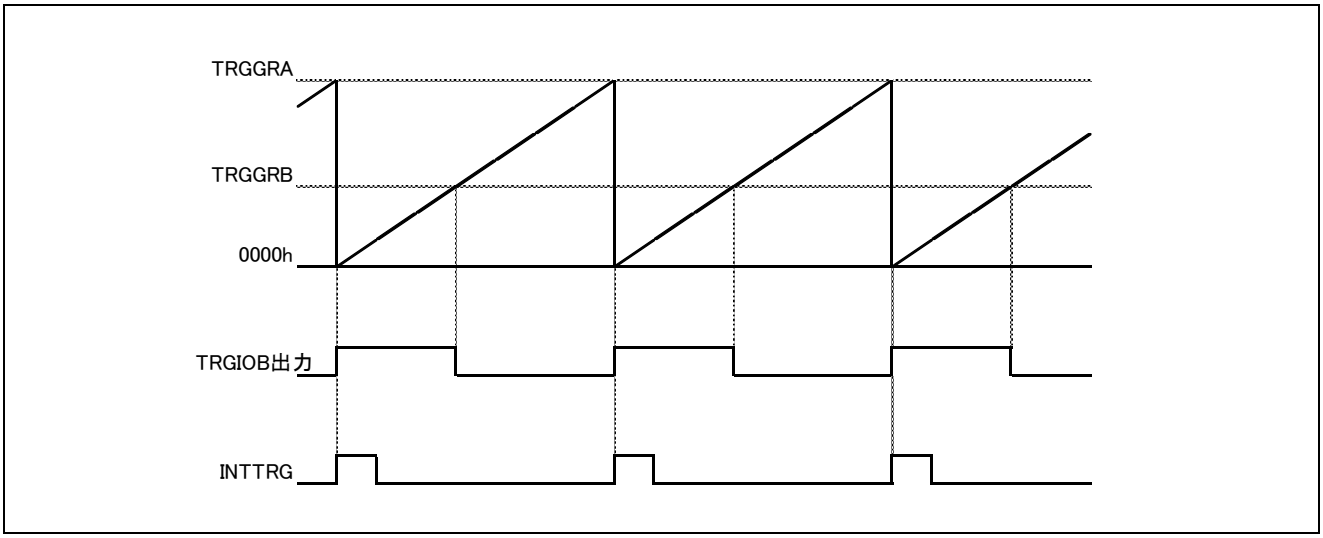
```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start the TMRG module operation */
    R_TMRG0_Start();

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

使用例(PWM モード)

指定した周期/デューティの PWM 出力を行う

[波形例]



[GUI 設定例]

タイマ RG				使用する
	TMRG			使用する
			機能	PWM モード
			カウントソース設定	内部クロック
			内部クロック設定	自動
			カウンタクリア	TRGGRA コンペアー致でクリア
			レジスタ機能設定 (TRGGRC)	ジェネラルレジスタ
			レジスタ機能設定 (TRGGRD)	ジェネラルレジスタ
			周期	100μs (実際の値：100)
			デューティ	50(%) (実際の値：50)
			TRGGRA コンペアー致割り込み許可	使用する
			TRGGRB コンペアー致割り込み許可	使用しない
			TRG オーバフロー割り込み許可	使用しない
			INTTRG 優先順位	レベル 3(低優先順位)

[API 設定例]

r_cg_main.c

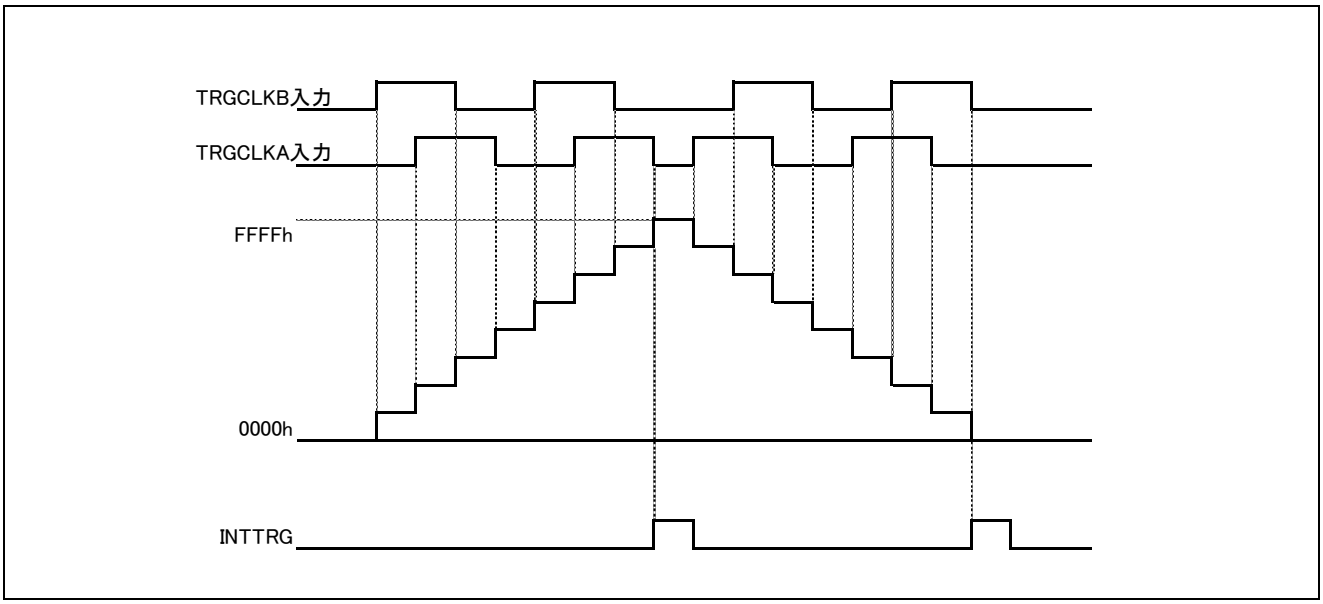
```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start the TMRG module operation */
    R_TMRG0_Start();

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

使用例(位相計数モード)

TRGCLKA、TRGCLKB 端子からの外部入力信号の位相差を検出し、TRG レジスタをアップ/ダウンカウントさせ、オーバフロー回数とアンダフロー回数を数える。

[波形例]



[GUI 設定例]

タイマ RG				使用する
	TMRG			使用する
			機能	位相計数モード
			初期計数	0
			カウンタクリア	クリア禁止
			CNTEN0	使用する
			CNTEN1	使用する
			CNTEN2	使用する
			CNTEN3	使用する
			CNTEN4	使用する
			CNTEN5	使用する
			CNTEN6	使用する
			CNTEN7	使用する
			TRG オーバフロー割り込み許可	使用する
			TRG アンダフロー割り込み許可	使用する
			INTTRG 優先順位	レベル 3(低優先順位)

[API 設定例]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start the TMRG module operation */
    R_TMRG0_Start();

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

r_cg_tmrg_user.c

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

static void __near r_tmrg0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    uint8_t temp_trg = 0U;

    /* === Count number of overflow or under flow === */
    /* Mask TRGSR resister to check overflow or underflow occurred */
    temp_trg = TRGSR & 0x0CU;

    if (temp_trg == 0x08U)
    {
        /* --- Count up number of overflow --- */
        inttrg_over_cnt++;

        /* --- Clear overflow Flag --- */
        TRGSR &= 0x07U;
    }
    else
    {
        /* --- Count up number of underflow --- */
        inttrg_under_cnt++;

        /* --- Clear under flow Flag --- */
        TRGSR &= 0x0BU;
    }
    /* End user code. Do not edit comment generated here */
}
```

3.3.9 タイマ RX

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

表 3.9 タイマ RX 用 API 関数

API 関数名	機能概要
R_TMRX_Create	16 ビット・タイマ RX を制御するうえで必要となる初期化処理を行います。
R_TMRX_Create_UserInit	16 ビット・タイマ RX に関するユーザ独自の初期化処理を行います。
r_tmrx_interrupt	タイマ割り込みの発生に伴う処理を行います。
R_TMRX_Start	16 ビット・タイマ RX のカウント処理を開始します。
R_TMRX_Stop	16 ビット・タイマ RX のカウント処理を終了します。
R_TMRX_Set_PowerOff	16 ビット・タイマ RX に対するクロック供給を停止します。
R_TMRX_Get_BufferValue	16 ビット・タイマ RX の TRX レジスタのバッファ・レジスタを読み出します。

R_TMRX_Create

16 ビット・タイマ RX を制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void        R_TMRX_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_TMRX_Create_UserInit

16 ビット・タイマ RX 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_TMRX_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_TMRX_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_tmrx_interrupt

タイマ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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_tmrx_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_tmrx_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_TMRX_Start

16 ビット・タイマ RX のカウント処理を開始します。

[指定形式]

```
void    R_TMRX_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_TMRX_Stop

16 ビット・タイマ RX のカウント処理を終了します。

[指定形式]

```
void    R_TMRX_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_TMRX_Set_PowerOff

16 ビット・タイマ RX に対するクロック供給を停止します。

備考　本 API 関数を呼び出しにより、16 ビット・タイマ RX はリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

```
void    R_TMR_RX_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

R_TMRX_Get_BufferValue

16 ビット・タイマ RX の TRX レジスタのバッファ・レジスタを読み出します。

[指定形式]

```
#include   “r_cg_macrodriver.h”
void       R_TMRX_Get_BufferValue ( uint32_t * const value );
```

[引数]

I/O	引数	説明
O	unit32_t * const *value*;	TRX レジスタのバッファ・レジスタの値を格納する領域へのポインタ

[戻り値]

なし

使用例

カウンタオーバフロー時にタイマを停止させる

[GUI 設定例]

タイマ RX				使用する
	TMRX			使用する
			タイマ RX 動作設定	使用する
			カウントソース設定	fCLK
			カウント開始要因設定	ソフトウェア
			ソフトウェアでのリセットイネーブル信号設定	ソフトウェアでカウント・リセット許可
			コンパレータ 1 のトリガ設定	タイマ RX カウンタのカウント値をタイマ RX カウント・バッファ・レジスタへ転送、タイマ RX カウント値を 0000H にし、カウント継続
			TRX のオーバフロー時に割り込みを許可 (INTTRX)	使用する
			優先順位	レベル 3(低優先順位)

[API 設定例]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start TMRX counter */
    R_TMRX_Start();

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

r_cg_tmrx_user.c

```
static void __near r_tmrx_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop TMRX counter */
    R_TMRX_Stop();
    /* End user code. Do not edit comment generated here */
}
```

照

3.3.10 16 ビット・タイマ KB

以下に、コード生成ツールが 16 ビット・タイマ KB 用として出力する API 関数の一覧を示します。

表 3.10 16 ビット・タイマ KB 用 API 関数

API 関数名	機能概要
R_TMR_KB_Create	16 ビット・タイマ KB を制御するうえで必要となる初期化処理を行います。
R_TMR_KB_Create_UserInit	16 ビット・タイマ KB に関するユーザ独自の初期化処理を行います。
r_tmr_kbm_interrupt	タイマ割り込みの発生に伴う処理を行います。
R_TMR_KBm_Start	16 ビット・タイマ KB のカウント処理を開始します。
R_TMR_KBm_Stop	16 ビット・タイマ KB のカウント処理を終了します。
R_TMR_KBm_Set_PowerOff	16 ビット・タイマ KB に対するクロック供給を停止します。
R_TMR_KBmn_ForcedOutput_Start	強制出力停止機能に使用するトリガ信号の入力を許可します。
R_TMR_KBmn_ForcedOutput_Stop	強制出力停止機能に使用するトリガ信号の入力を禁止します。
R_TMR_KBm_BatchOverwriteRequestOn	コンペア・レジスタの一斉書き換えを許可します。
R_TMR_KBm_ForcedOutput_mn_Start	強制出力停止機能に使用するトリガ信号の入力を許可します。
R_TMR_KBm_ForcedOutput_mn_Stop	強制出力停止機能に使用するトリガ信号の入力を禁止します。
R_TMR_KBm_Reset	16 ビット・タイマ KB をリセットします。

R_TMR_KB_Create

16 ビット・タイマ KB を制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_TMR_KB_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_TMR_KB_Create_UserInit

16 ビット・タイマ KB 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_KB_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_KB_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_tmr_kb*m*_interrupt

タイマ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]

CA78K0R コンパイラの場合

```
__interrupt static void    r_tmr_kbm_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void    __near r_tmr_kbm_interrupt ( void );
```

備考　　*m* はユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_TMR_KB*m*_Start

16 ビット・タイマ KB*m* のカウント処理を開始します。

[指定形式]

```
void    R_TMR_KBm_Start ( void );
```

備考　　*m* は、ユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_TMR_KB*m*_Stop

16 ビット・タイマ KB*m*のカウント処理を終了します。

[指定形式]

```
void    R_TMR_KBm_Stop ( void );
```

備考　　*m* は、ユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_TMR_KB*m*_Set_PowerOff

16 ビット・タイマ KB*m* に対するクロック供給を停止します。

備考　本 API 関数を呼び出しにより、16 ビット・タイマ KB*m* はリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

```
void    R_TMR_KBm_Set_PowerOff ( void );
```

備考　*m* は、ユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_TMR_KB*mn*_ForcedOutput_Start

強制出力停止機能に使用するトリガ信号の入力を許可します。

[指定形式]

```
void    R_TMR_KBmn_ForcedOurput_Start ( void );
```

備考　　*m* はユニット番号を、*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_KB*mn*_ForcedOutput_Stop

強制出力停止機能に使用するトリガ信号の入力を禁止します。

[指定形式]

```
void    R_TMR_KBmn_ForcedOutput_Stop ( void );
```

備考　　*m* はユニット番号を、*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_KB*m*_BatchOverwriteRequestOn

コンペア・レジスタの一斉書き換えを許可します。

備考　コンペア・レジスタの内容を一斉に書き換えるタイミングは、本 API 関数を呼び出したのち、カウント値とコンペア・レジスタに設定された値が一致した際、または外部トリガが発生した際となります。

[指定形式]

```
void    R_TMR_KBm_BatchOverwritRequestOn ( void );
```

[引数]

なし

[戻り値]

なし

R_TMR_KB*m*_ForcedOutput_*mn*_Start

強制出力停止機能に使用するトリガ信号の入力を許可します。

[指定形式]

```
void    R_TMR_KBm_ForcedOurput_mn_Start ( void );
```

備考　　*m* はユニット番号を、*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_KB*m*_ForcedOutput_*mn*_Stop

強制出力停止機能に使用するトリガ信号の入力を禁止します。

[指定形式]

```
void    R_TMR_KBm_ForcedOutput_mn_Stop ( void );
```

備考　　*m* はユニット番号を、*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_KB*m*_Reset

16 ビット・タイマ KB をリセットします。

[指定形式]

void	R_TMR_KB*m*_Reset (void);

備考 *m* は、ユニット番号を意味します。

[引数]

なし

[戻り値]

なし

使用例

ワンショットタイマとして使用する

[GUI 設定例]

タイマ				使用する
	TMKB			使用する
			A/D トリガ設定	タイマ KB0 のトリガ要因
		TMKB0		
			TMKB0	単体動作モード
		TMKB_STANDALONE_0		
			TKBO00	使用しない
			TKBO01	使用しない
			周期設定	50ms (実際の値：50)
			TKBO00 デューティ	0(%) (実際の値：0%)
			TKBO01 デューティ	0(%) (実際の値：0%)
			TKBO01 ディレイ	0μs (実際の値：0)
			トリガ入力を使用	使用しない
			TKC00 による出力のゲート機能	使用しない
			TKC01 による出力のゲート機能	使用しない
			A/D 変換スタート・タイミング設定	0μs (実際の値：0)
			タイマ・チャネル 0 のカウント完了で割り込み発生(INTTMKB0)	使用する
			優先順位 (INTTMKB0)	低
			ソフト・スタート機能設定 TKBO00	Unused
			ソフト・スタート機能設定 TKBO01	Unused
			ディザリング機能設定 TKBO00	Unused
			ディザリング機能設定 TKBO01	Unused
			強制出力停止機能設定 TKBO00	Unused
			強制出力停止機能設定 TKBO01	Unused

[API 設定例]

r_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start TMKB0 counter */
    R_TMR_KB0_Start();

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

r_cg_timer_user.c

```
static void __near r_tmr_kb0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop TMKB0 counter */
    R_TMR_KB0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

3.3.11 16 ビット・タイマ KC0

以下に、コード生成ツールが 16 ビット・タイマ KC0 用として出力する API 関数の一覧を示します。

表 3.11 16 ビット・タイマ KC0 用 API 関数

API 関数名	機能概要
R_TMR_KC0_Create	16 ビット・タイマ KC0 を制御するうえで必要となる初期化処理を行います。
R_TMR_KC0_Create_UserInit	16 ビット・タイマ KC0 に関するユーザ独自の初期化処理を行います。
r_tmr_kc0_interrupt	タイマ割り込みの発生に伴う処理を行います。
R_TMR_KC0_Start	16 ビット・タイマ KC0 のカウント処理を開始します。
R_TMR_KC0_Stop	16 ビット・タイマ KC0 のカウント処理を終了します。
R_TMR_KC0_Set_PowerOff	16 ビット・タイマ KC0 に対するクロック供給を停止します。

R_TMR_KC0_Create

16 ビット・タイマ KC0 を制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_TMR_KC0_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_TMR_KC0_Create_UserInit

16 ビット・タイマ KC0 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_KC0_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_KC0_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_tmr_kc0_interrupt

タイマ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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_tmr_kc0_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_tmr_kc0_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_TMR_KC0_Start

16 ビット・タイマ KC0 のカウント処理を開始します。

[指定形式]

```
void    R_TMR_KC0_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_TMR_KC0_Stop

16 ビット・タイマ KC0 のカウント処理を終了します。

[指定形式]

```
void    R_TMR_KC0_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_TMR_KC0_Set_PowerOff

16 ビット・タイマ KC0 に対するクロック供給を停止します。

備考　本 API 関数を呼び出しにより、16 ビット・タイマ KC0 はリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

```
void    R_TMR_KC0_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

使用例

ワンショットタイマとして使用する

[GUI 設定例]

タイマ				使用する
	TMKC			使用する
			動作モード設定	単体動作モード
		TMKC0		
			TKCO00	使用しない
			TKCO01	使用しない
			TKCO02	使用しない
			TKCO03	使用しない
			TKCO04	使用しない
			TKCO05	使用しない
			周期設定	50(実際の値：50)
			TKCO00 デューティ	0(%)(実際の値：0%)
			TKCO01 デューティ	0(%)(実際の値：0%)
			TKCO02 デューティ	0(%)(実際の値：0%)
			TKCO03 デューティ	0(%)(実際の値：0%)
			TKCO04 デューティ	0(%)(実際の値：0%)
			TKCO05 デューティ	0(%)(実際の値：0%)
			タイマ・チャネル 0 のカウント完了で割り込み発生(INTTMKC0)	使用する
			優先順位(INTTMKC0)	低

[API 設定例]

r_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start TMKC channel 0 counter */
    R_TMR_KC0_Start();

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

r_cg_timer_user.c

```
static void __near r_tmr_kc0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop TMKC channel 0 counter */
    R_TMR_KC0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

3.3.12 16 ビット・タイマ KB2

以下に、コード生成ツールが 16 ビット・タイマ KB2 用として出力する API 関数の一覧を示します。

表 3.12 16 ビット・タイマ KB2 用 API 関数

API 関数名	機能概要
R_KB2m_Create	16 ビット・タイマ KB2 を制御するうえで必要となる初期化処理を行います。
R_KB2m_Create_UserInit	16 ビット・タイマ KB2 に関するユーザ独自の初期化処理を行います。
r_kb2m_interrupt	タイマ割り込み INTTKB2*m* の発生に伴う処理を行います。
R_KB2m_Start	16 ビット・タイマ KB2 のカウント処理を開始します。
R_KB2m_Stop	16 ビット・タイマ KB2 のカウント処理を終了します。
R_KB2m_Set_PowerOff	16 ビット・タイマ KB2 に対するクロック供給を停止します。
R_KB2m_Simultaneous_Start	同時スタート&ストップ・モードを開始します。
R_KB2m_Simultaneous_Stop	同時スタート&ストップ・モードを終了します。
R_KB2m_Synchronous_Start	タイマ・スタート&クリア・モードを開始します。
R_KB2m_Synchronous_Stop	タイマ・スタート&クリア・モードを終了します。
R_KB2m_TKBOn0_Forced_Output_Stop_Function1_Start	タイマ出力 TKBO*n*0 に対する強制出力停止機能 1 を開始します。
R_KB2m_TKBOn0_Forced_Output_Stop_Function1_Stop	タイマ出力 TKBO*n*0 に対する強制出力停止機能 1 を終了します。
R_KB2m_TKBOn1_Forced_Output_Stop_Function1_Start	タイマ出力 TKBO*n*1 に対する強制出力停止機能 2 を開始します。
R_KB2m_TKBOn1_Forced_Output_Stop_Function1_Stop	タイマ出力 TKBO*n*1 に対する強制出力停止機能 2 を終了します。
R_KB2m_TKBOn0_DitheringFunction_Start	タイマ出力 TKBO*n*0 に対するディザリング機能を開始します。
R_KB2m_TKBOn0_DitheringFunction_Stop	タイマ出力 TKBO*n*0 に対するディザリング機能を終了します。
R_KB2m_TKBOn1_DitheringFunction_Start	タイマ出力 TKBO*n*1 に対するディザリング機能を開始します。
R_KB2m_TKBOn1_DitheringFunction_Stop	タイマ出力 TKBO*n*1 に対するディザリング機能を終了します。
R_KB2m_TKBOn0_SmoothStartFunction_Start	タイマ出力 TKBO*n*0 に対するソフト・スタート機能を開始します。
R_KB2m_TKBOn0_SmoothStartFunction_Stop	タイマ出力 TKBO*n*0 に対するソフト・スタート機能を終了します。
R_KB2m_TKBOn1_SmoothStartFunction_Start	タイマ出力 TKBO*n*1 に対するソフト・スタート機能を開始します。
R_KB2m_TKBOn1_SmoothStartFunction_Stop	タイマ出力 TKBO*n*1 に対するソフト・スタート機能を終了します。
R_KB2m_BatchOverwriteRequestOn	コンペア・レジスタの一斉書き換えを許可します。

R_KB2*m*_Create

16 ビット・タイマ KB2 を制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_KB2m_Create ( void );
```

備考　*m* はユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_KB2*m*_Create_UserInit

16 ビット・タイマ KB2 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_KB2m_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_KB2*m*_Create_UserInit (void);

備考　*m* はユニット番号を意味します。

[引数]

なし

[戻り値]

なし

r_kb2*m*_interrupt

タイマ割り込み INTTKB2*m* の発生に伴う処理を行います。

備考　本 API 関数は、タイマ割り込み INTTKB2*m* 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]

CA78K0R コンパイラの場合

```
__interrupt static void    r_kb2m_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void    __near r_kb2m_interrupt ( void );
```

備考　*m* はユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_KB2*m*_Start

16 ビット・タイマ KB2 のカウント処理を開始します。

[指定形式]

```
void    R_KB2m_Start ( void );
```

備考　　*m* は、ユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_KB2*m*_Stop

16 ビット・タイマ KB2 のカウント処理を終了します。

[指定形式]

```
void    R_KB2m_Stop ( void );
```

備考　　*m* は、ユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_KB2*m*_Set_PowerOff

16 ビット・タイマ KB2 に対するクロック供給を停止します。

[指定形式]

```
void    R_KB2m_Set_PowerOff ( void );
```

備考　　*m* は、ユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_KB2*m*_Simultaneous_Start

同時スタート&ストップ・モードを開始します。

[指定形式]

```
void    R_KB2m_Simultaneous_Start ( void );
```

備考　　*m* は、ユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_KB2*m*_Simultaneous_Stop

同時スタート&ストップ・モードを終了します。

[指定形式]

```
void    R_KB2m_Simultaneous_Stop ( void );
```

備考　　*m* は、ユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_KB2*m*_Synchronous_Start

タイマ・スタート&クリア・モードを開始します。

[指定形式]

```
void    R_KB2m_Synchronous_Start ( void );
```

備考　　*m* は、ユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_KB2*m*_Synchronous_Stop

タイマ・スタート&クリア・モードを終了します。

[指定形式]

```
void    R_KB2m_Synchronous_Stop ( void );
```

備考　　*m* は、ユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_KB2*m*_TKBO*n*0_Forced_Output_Stop_Function1_Start

タイマ出力 TKBO*n*0 に対する強制出力停止機能 1 を開始します。

[指定形式]

```
void    R_KB2m_TKBOn0_Forced_Ourput_Stop_Function1_Start ( void );
```

備考　　*m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_KB2*m*_TKBO*n*0_Forced_Output_Stop_Function1_Stop

タイマ出力 TKBO*n*0 に対する強制出力停止機能 1 を終了します。

[指定形式]

```
void    R_KB2m_TKBOn0_Forced_Output_Stop_Function1_Stop ( void );
```

備考　　*m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_KB2*m*_TKBO*n*1_Forced_Output_Stop_Function1_Start

タイマ出力 TKBO*n*1 に対する強制出力停止機能 2 を開始します。

[指定形式]

```
void    R_KB2m_TKBOn1_Forced_Ourput_Stop_Function1_Start ( void );
```

備考　　*m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_KB2*m*_TKBO*n*1_Forced_Output_Stop_Function1_Stop

タイマ出力 TKBO*n*1 に対する強制出力停止機能 2 を終了します。

[指定形式]

```
void    R_KB2m_TKBOn1_Forced_Output_Stop_Function1_Stop ( void );
```

備考　　*m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_KB2*m*_TKBO*n*0_DitheringFunction_Start

タイマ出力 TKBO*n*0 に対するディザリング機能を開始します。

[指定形式]

```
void    R_KB2m_TKBOn0_DitheringFunction_Start ( void );
```

備考　　*m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_KB2*m*_TKBO*n*0_DitheringFunction_Stop

タイマ出力 TKBO*n*0 に対するディザリング機能を終了します。

[指定形式]

```
void    R_KB2m_TKBOn0_DitheringFunction_Stop ( void );
```

備考　　*m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_KB2*m*_TKBO*n*1_DitheringFunction_Start

タイマ出力 TKBO*n*1 に対するディザリング機能を開始します。

[指定形式]

```
void    R_KB2m_TKBOn1_DitheringFunction_Start ( void );
```

備考　　*m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_KB2*m*_TKBO*n*1_DitheringFunction_Stop

タイマ出力 TKBO*n*1 に対するディザリング機能を終了します。

[指定形式]

```
void    R_KB2m_TKBOn1_DitheringFunction_Stop ( void );
```

備考　　*m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_KB2*m*_TKBO*n*0_SmoothStartFunction_Start

タイマ出力 TKBO*n*0 に対するソフト・スタート機能を開始します。

[指定形式]

```
void    R_KB2m_TKBOn0_SmoothStartFunction_Start ( void );
```

備考　　*m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_KB2*m*_TKBO*n*0_SmoothStartFunction_Stop

タイマ出力 TKBO*n*0 に対するソフト・スタート機能を終了します。

[指定形式]

```
void    R_KB2m_TKBOn0_SmoothStartFunction_Stop ( void );
```

備考　　*m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_KB2*m*_TKBO*n*1_SmoothStartFunction_Start

タイマ出力 TKBO*n*1 に対するソフト・スタート機能を開始します。

[指定形式]

```
void    R_KB2m_TKBOn1_SmoothStartFunction_Start ( void );
```

備考　　*m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_KB2*m*_TKBO*n*1_SmoothStartFunction_Stop

タイマ出力 TKBO*n*1 に対するソフト・スタート機能を終了します。

[指定形式]

```
void    R_KB2m_TKBOn1_SmoothStartFunction_Stop ( void );
```

備考　　*m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_KB2*m*_BatchOverwriteRequestOn

コンペア・レジスタの一斉書き換えを許可します。

備考 コンペア・レジスタの内容を一斉に書き換えるタイミングは、本 API 関数を呼び出したのち、カウント値とコンペア・レジスタに設定された値が一致した際、または外部トリガが発生した際となります。

[指定形式]

void	R_KB2*m*_BatchOverwritRequestOn (void);

備考 *m* はユニット番号を意味します。

[引数]

なし

[戻り値]

なし

使用例

ワンショットタイマとして使用する

[GUI 設定例]

タイマ KB2				使用する
	KB2			使用する
		TKB20		使用する
			TKB20	単体動作モード(TKBCR00 による周期制御)
			クロック設定	TKBTCK0 を選択
			動作クロック設定	fCLK
			パルス周期	1ms(実際の値：1)
			デューティ(TKBO00 出力)	0(%)(実際の値：0)
			ディレイ(TKBO01 出力)	0(%)(実際の値：0)
			デューティ(TKBO01 出力)	0(%)(実際の値：0)
			TKBO00 の PWM 出力ソフトスタート機能設定	使用しない
			TKBO01 の PWM 出力ソフトスタート機能設定	使用しない
			TKBO00 の PWM 出力ディザリング機能設定	使用しない
			TKBO01 の PWM 出力ディザリング機能設定	使用しない
			TKBTGCR0 値	0
			TKBO00 出力設定	禁止
			TKBO01 出力設定	禁止
			タイマ KB20 のカウント完了割り込み(INTTKB20)	使用する
			優先順位	低
			強制出力停止機能設定(TKBO00)	禁止
			強制出力停止機能設定(TKBO01)	禁止

[API 設定例]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start KB20 module operation */
    R_KB20_Start();

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

r_cg_kb2_user.c

```
static void __near r_kb20_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop KB20 module operation */
    R_KB20_Stop();
    /* End user code. Do not edit comment generated here */
}
```

3.3.13 リアルタイム・クロック

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

表 3.13 リアルタイム・クロック用 API 関数 (1)

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_interrupt	リアルタイム・クロック割り込み INTRTC の発生に伴う処理を行います。
R_RTC_Start	リアルタイム・クロック（年，月，曜日，日，時，分，秒）のカウントを開始します。
R_RTC_Stop	リアルタイム・クロック（年，月，曜日，日，時，分，秒）のカウントを終了します。
R_RTC_Set_PowerOff	リアルタイム・クロックに対するクロック供給を停止します。
R_RTC_Set_HourSystem	リアルタイム・クロックの時間制（12 時間制，24 時間制）を設定します。
R_RTC_Set_CounterValue	リアルタイム・クロックにカウント値を設定します。
R_RTC_Set_CalendarCounterValue	リアルタイム・クロックにカウント値を設定します。（カレンダモード設定時）
R_RTC_Set_BinaryCounterValue	リアルタイム・クロックにカウント値を設定します。（バイナリモード設定時）
R_RTC_Get_CounterValue	リアルタイム・クロックにカウント値を読み出します。
R_RTC_Get_CalendarCounterValue	リアルタイム・クロックにカウント値を読み出します。（カレンダモード設定時）
R_RTC_Get_BinaryCounterValue	リアルタイム・クロックにカウント値を読み出します。（バイナリモード設定時）
R_RTC_Set_ConstPeriodInterruptOn	割り込み INTRTC の発生周期を設定したのち、定周期割り込み機能を開始します。
R_RTC_Set_ConstPeriodInterruptOff	定周期割り込み機能を終了します。
r_rtc_callback_constperiod	定周期割り込み INTRTC の発生に伴う処理を行います。
R_RTC_Set_AlarmOn	アラーム割り込み機能を開始します。
R_RTC_Set_CalendarAlarmOn	アラーム割り込み機能を開始します。（カレンダモード設定時）
R_RTC_Set_BinaryAlarmOn	アラーム割り込み機能を開始します。（バイナリモード設定時）
R_RTC_Set_AlarmOff	アラーム割り込み機能を終了します。
R_RTC_Set_AlarmValue	アラームの発生条件（曜日，時，分）を設定します。
R_RTC_Get_CalenderAlarmValue	アラームの発生条件（年，月，曜日，日，時，分，秒）を設定します。（カレンダモード設定時）
R_RTC_Set_BinaryAlarmValue	アラームの発生条件を設定します。（バイナリモード設定時）
R_RTC_Get_AlarmValue	アラームの発生条件（曜日，時，分）を読み出します。

表 3.14 リアルタイム・クロック用 API 関数 (2)

API 関数名	機能概要
R_RTC_Get_CalenderAlarmValue	アラームの発生条件（年，月，曜日，日，時，分，秒）を読み出します。（カレンダモード設定時）
R_RTC_Get_BinaryAlarmValue	アラームの発生条件を読み出します。（バイナリモード設定時）
r_rtc_callback_alarm	アラーム割り込み INTRTC の発生に伴う処理を行います。
R_RTC_Set_RTC1HZOn	RTC1HZ 端子に対する補正クロック（1Hz）の出力を許可します。
R_RTC_Set_RTC1HZOff	RTC1HZ 端子に対する補正クロック（1Hz）の出力を禁止します。
R_RTC_Set_RTCOUTOn	RTCOUT の出力を許可します。
R_RTC_Set_RTCOUTOff	RTCOUT の出力を禁止します。
r_rtc_alarminterrupt	アラーム割り込み INTRTCALM の発生に伴う処理を行います。
r_rtc_periodinterrupt	周期割り込み INTRTCPRD の発生に伴う処理を行います。
r_rtc_callback_periodic	定周期割り込み INTRTC の発生に伴う処理を行います。

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。

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

[指定形式]

```
void    R_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_interrupt

リアルタイム・クロック割り込み INTRTC の発生に伴う処理を行います。

備考　本 API 関数は、リアルタイム・クロック割り込み INTRTC 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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_rtc_interrupt ( void );
```

CC-RL コンパイラの場合

```
Static void     __near r_rtc_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_RTC_Start

リアルタイム・クロック（年，月，曜日，日，時，分，秒）の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

リアルタイム・クロック（年，月，曜日，日，時，分，秒）の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_Set_PowerOff

リアルタイム・クロックに対するクロック供給を停止します。

備考 1. 本 API 関数を呼び出しにより、リアルタイム・クロックはリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

備考 2. 本 API 関数では、周辺イネーブル・レジスタ n の RTCEN ビットを操作することにより、リアルタイム・クロックに対するクロック供給の停止を実現しています。
このため、本 API 関数の呼び出しを行った際には、RTCEN ビットを共用しているほかの周辺機能（インターバル・タイマなど）に対するクロック供給も停止することになります。

[指定形式]

```
void        R_RTC_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

R_RTC_Set_HourSystem

リアルタイム・クロックの時間制（12 時間制，24 時間制）を設定します。

[指定形式]

```
#include   “r_cg_macrodriver.h”
#include   “r_cg_rtc.h”
void       R_RTC_Set_HourSystem ( rtc_hour_system_t hour_system );
```

[引数]

I/O	引数	説明
I	rtc_hour_system_t *hour_system*;	時間制の種類 HOUR12 ：12 時間制 HOUR24 ：24 時間制

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_BUSY1	カウント処理を実行中（設定変更前）
MD_BUSY2	カウント処理を実行中（設定変更後）
MD_ARGERROR	引数の指定が不正

備考　MD_BUSY1、または MD_BUSY2 が返却される場合は、カウンタの動作が停止している、またはカウンタの動作開始待ち時間が短いことに起因している可能性があるため、ヘッダ・ファイル r_cg_rtc.h で定義されているマクロ RTC_WAITTIME の値を大きくしてください。

R_RTC_Set_CounterValue

リアルタイム・クロックにカウント値を設定します。

備考 カウンタ動作中(RTCE = 1) に本関数で SEC, MIN, HOUR, WEEK, DAY, MONTH, YEAR レジスタを書き換える場合は、INTRTC を割り込みマスク・フラグ・レジスタで割り込み処理禁止にしてからコールしてください。また、書き換え後に WAFG フラグ、RIFG フラグ、RTCIF フラグをクリアしてください。

[指定形式]

```
#include   “r_cg_macrodriver.h”
#include   “r_cg_rtc.h”
void       R_RTC_Set_CounterValue ( rtc_counter_value_t counter_write_val );
```

[引数]

I/O	引数	説明
I	rtc_counter_value_t *counter_write_val*;	カウント値

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

```
typedef struct {
    uint8_t sec;        /*  秒  */
    uint8_t min;        /*  分  */
    uint8_t hour;       /*  時  */
    uint8_t day;        /*  日  */
    uint8_t week;       /*  曜日（ 0 ：日曜日，  6 ：土曜日）  */
    uint8_t month;      /*  月  */
    uint16_t year;      /*  年  */
} rtc_counter_value_t;
```

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_BUSY1	カウント処理を実行中（設定変更前）
MD_BUSY2	カウント処理を実行中（設定変更後）

備考 MD_BUSY1、または MD_BUSY2 が返却される場合は、カウンタの動作が停止している、またはカウンタの動作開始待ち時間が短いことに起因している可能性があるため、ヘッダ・ファイル r_cg_rtc.h で定義されているマクロ RTC_WAITTIME の値を大きくしてください。

R_RTC_Set_CalendarCounterValue

リアルタイム・クロックにカウント値を設定します。（カレンダモード設定時）

[指定形式]

```
#include   “r_cg_macrodriver.h”
#include   “r_cg_rtc.h”
void       R_RTC_Set_CalendarCounterValue ( rtc_counter_value_t counter_write_val );
```

[引数]

I/O	引数	説明
I	rtc_counter_value_t *counter_write_val*;	カウント値（年，月，日，曜日，時，分，秒）

備考　カウント値 rtc_counter_value_t についての詳細は、R_RTC_Set_CounterValue を参照してください。

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_BUSY1	カウント処理を実行中（設定変更前）

備考　MD_BUSY1 が返却される場合は、カウンタの動作開始待ち時間が短いことに起因している可能性があるため、ヘッダ・ファイル r_cg_rtc.h で定義されているマクロ RTC_WAITTIME の値を大きくしてください。

R_RTC_Set_BinaryCounterValue

リアルタイム・クロックにカウント値を設定します。（バイナリモード設定時）

[指定形式]

```
#include   “r_cg_macrodriver.h”
#include   “r_cg_rtc.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*;	カウント値

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_BUSY1	カウント処理を実行中（設定変更前）

備考　MD_BUSY1 が返却される場合は、カウンタの動作開始待ち時間が短いことに起因している可能性があるため、ヘッダ・ファイル r_cg_rtc.h で定義されているマクロ RTC_WAITTIME の値を大きくしてください。

R_RTC_Get_CounterValue

リアルタイム・クロックのカウント値を読み出します。

[指定形式]

```
#include   “r_cg_macrodriver.h”
#include   “r_cg_rtc.h”
void       R_RTC_Get_CounterValue ( rtc_counter_value_t * const counter_read_val );
```

[引数]

I/O	引数	説明
O	rtc_counter_value_t * const *counter_read_val*;	読み出したカウント値を格納する構造体へのポインタ

備考　カウント値 rtc_counter_value_t についての詳細は、R_RTC_Set_CounterValue を参照してください。

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_BUSY1	カウント処理を実行中（設定変更前）
MD_BUSY2	カウント処理を実行中（設定変更後）

備考　MD_BUSY1、または MD_BUSY2 が返却される場合は、カウンタの動作が停止している、またはカウンタの動作開始待ち時間が短いことに起因している可能性があるため、ヘッダ・ファイル r_cg_rtc.h で定義されているマクロ RTC_WAITTIME の値を大きくしてください。

R_RTC_Get_CalendarCounterValue

リアルタイム・クロックのカウント値を読み出します。（カレンダモード設定時）

[指定形式]

```
#include   “r_cg_macrodriver.h”
#include   “r_cg_rtc.h”
void       R_RTC_Get_CalendarCounterValue ( rtc_counter_value_t * const counter_read_val );
```

[引数]

I/O	引数	説明
O	rtc_counter_value_t * const *counter_read_val*;	読み出したカウント値を格納する構造体へのポインタ

備考　　カウント値 rtc_counter_value_t についての詳細は、R_RTC_Set_CounterValue を参照してください。

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR	読み出し失敗

R_RTC_Get_BinaryCounterValue

リアルタイム・クロックのカウント値を読み出します。（バイナリモード設定時）

[指定形式]

```
#include   “r_cg_macrodriver.h”
#include   “r_cg_rtc.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*;	読み出したカウント値を格納する構造体へのポインタ

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR	読み出し失敗

R_RTC_Set_ConstPeriodInterruptOn

割り込み INTRTC の発生周期を設定したのち、定周期割り込み機能を開始します。

[指定形式]

```
#include        “r_cg_macrodriver.h”
#include        “r_cg_rtc.h”
MD_STATUS     R_RTC_Set_ConstPeriodInterruptOn ( rtc_int_period_t period );
```

[引数]

I/O	引数	説明
I	rtc_int_period_t *period*;	割り込み INTRTC の発生周期 HALFSEC： 0.5 秒 ONESEC： 1 秒 ONEMIN： 1 分 ONEHOUR： 1 時間 ONEDAY： 1 日 ONEMONTH： 1 ヵ月

[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_RTC_Set_ConstPeriodInterruptOff

定周期割り込み機能を終了します。

[指定形式]

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

[引数]

なし

[戻り値]

なし

r_rtc_callback_constperiod

定周期割り込み INTRTC の発生に伴う処理を行います。

備考　本 API 関数は、定周期割り込み INTRTC に対応した割り込み処理 r_rtc_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_rtc_callback_constperiod ( void );
```

[引数]

なし

[戻り値]

なし

R_RTC_Set_AlarmOn

アラーム割り込み機能を開始します。

[指定形式]

```
void    R_RTC_Set_AlarmOn ( void );
```

[引数]

なし

[戻り値]

なし

R_RTC_Set_CalendarAlarmOn

アラーム割り込み機能を開始します。（カレンダモード設定時）

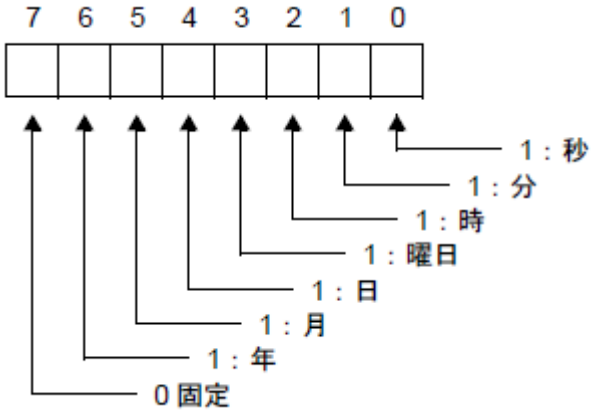
[指定形式]

```
#include   “r_cg_macrodriver.h”
#include   “r_cg_rtc.h”
void       R_RTC_Set_CalendarAlarmOn ( uint8_t enb_set );
```

[引数]

I/O	引数	説明
I	uint8_t *enb_set*;	アラームイネーブル

備考 1.　以下に、アラームイネーブル enb_set の各ビットに対する意味を示します。



[戻り値]

なし

R_RTC_Set_BinaryAlarmOn

アラーム割り込み機能を開始します。（バイナリモード設定時）

[指定形式]

```
void    R_RTC_Set_BinaryAlarm ( void );
```

[引数]

なし

[戻り値]

なし

R_RTC_Set_AlarmOff

アラーム割り込み機能を終了します。

[指定形式]

```
void    R_RTC_Set_AlarmOff ( void );
```

[引数]

なし

[戻り値]

なし

R_RTC_Set_AlarmValue

アラームの発生条件（曜日，時，分）を設定します。

[指定形式]

```
#include   “r_cg_macrodriver.h”
#include   “r_cg_rtc.h”
void       R_RTC_Set_AlarmValue ( rtc_alarm_value_t alarm_val );
```

[引数]

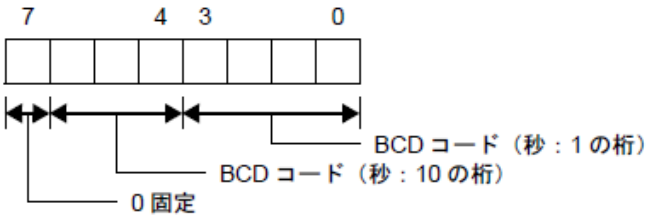
I/O	引数	説明
I	rtc_alarm_value_t *alarm_val*;	アラームの発生条件（曜日，時，分）

備考　以下に、アラーム発生条件 rtc_alarm_value_t の構成を示します。
（構成内容はデイバスによって胃異なります）

```
typedef struct {
    uint8_t alarmws;     /*  秒  */
    uint8_t alarmwm;     /*  分  */
    uint8_t alarmwh;     /*  時  */
    uint8_t alarmww;     /*  曜日（ 0 ：日曜日，  6 ：土曜日）  */
    uint8_t alarmwd;     /*  日  */
    uint8_t alarmwmt;    /*  月  */
    uint16_t alarmwy;    /*  年  */
} rtc_alarm_value_t;
```

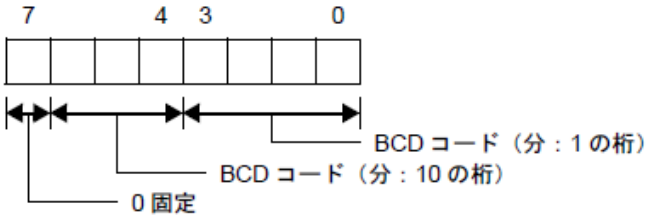
- alarmws（秒）

  以下に、構成メンバ alarmws の各ビットに対する意味を示します。



- alarmwm（分）

  以下に、構成メンバ alarmwm の各ビットに対する意味を示します。



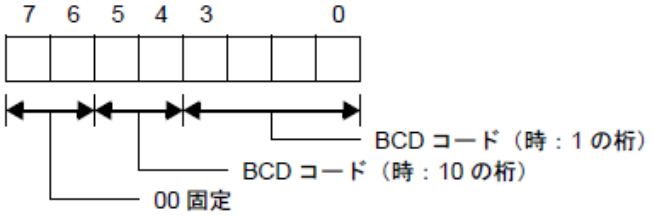
- alarmwh（時）

  以下に、構成メンバ alarmwh の各ビットに対する意味を示します。

  なお、ビット 5 は、リアルタイム・クロックが 12 時間制の場合、以下の意味となります。

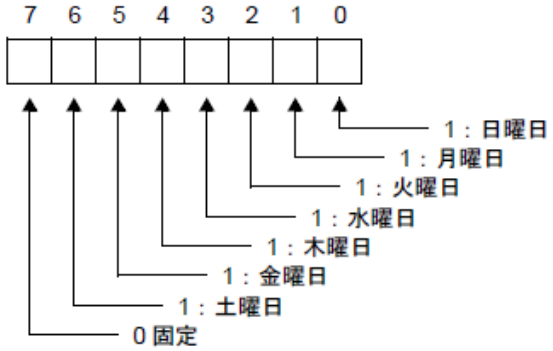
  0： 午前

  1： 午後



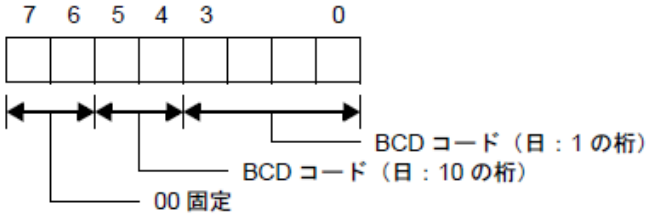
- alarmww（曜日）

  以下に、構成メンバ alarmww の各ビットに対する意味を示します。



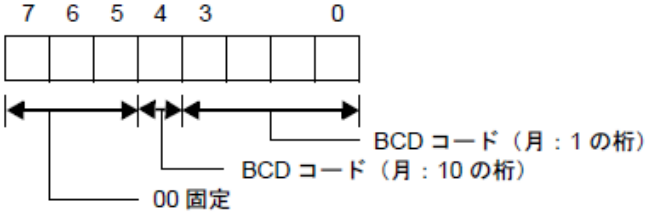
- alarmwd（日）

  以下に、構成メンバ alarmwd の各ビットに対する意味を示します。



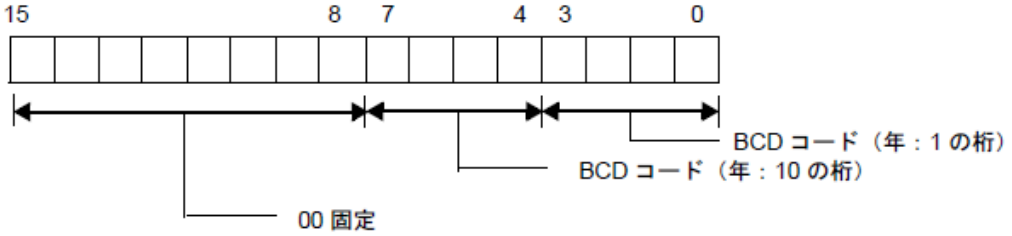
- alarmwmt（月）

  以下に、構成メンバ alarmwmt の各ビットに対する意味を示します。



- alarmwy（年）

以下に、構成メンバ alarmwy の各ビットに対する意味を示します。



[戻り値]

なし

R_RTC_Set_CalenderAlarmValue

アラームの発生条件（年，月，曜日，日，時，分，秒）を設定します。（カレンダモード設定時）

[指定形式]

```
#include   “r_cg_macrodriver.h”
#include   “r_cg_rtc.h”
void       R_RTC_Set_CalenderAlarmValue ( rtc_alarm_value_t alarm_val );
```

[引数]

I/O	引数	説明
I	rtc_alarm_value_t *alarm_val*;	アラームの発生条件（年，月，曜日，日，時，分，秒）

備考　以下に、アラーム発生条件 rtc_alarm_value_t についての詳細は、R_RTC_Set_AlarmValue を参照してください。

[戻り値]

なし

R_RTC_Set_BinaryAlarmValue

アラームの発生条件を設定します。（バイナリモード設定時）

[指定形式]

```
#include   “r_cg_macrodriver.h”
#include   “r_cg_rtc.h”
void       R_RTC_Set_BinaryAlarmValue ( uint32_t alarm_enable, uint32_t alarm_val );
```

[引数]

I/O	引数	説明
I	uint32_t *alarm_enable*;	アラームイネーブル （バイナリカウンタアラーム許可レジスタへ値を設定します）
I	uint32_t *alarm_val*;	アラームの発生条件（カウント値） （バイナリカウンタアラームレジスタへ値を設定します）

[戻り値]

なし

R_RTC_Get_AlarmValue

アラームの発生条件（曜日，時，分）を読み出します。

[指定形式]

```
#include   “r_cg_macrodriver.h”
#include   “r_cg_rtc.h”
void       R_RTC_Get_AlarmValue ( rtc_alarm_value_t * const alarm_val );
```

[引数]

I/O	引数	説明
I	rtc_alarm_value_t * const *alarm_val*;	読み出した発生条件を格納する構造体へのポインタ

備考　アラーム発生条件 rtc_alarm_value_t についての詳細は、R_RTC_Set_AlarmValue を参照してください。

[戻り値]

なし

R_RTC_Get_CalenderAlarmValue

アラームの発生条件（年，月，曜日，日，時，分，秒）を読み出します。（カレンダモード設定時）

[指定形式]

```
#include   “r_cg_macrodriver.h”
#include   “r_cg_rtc.h”
void       R_RTC_Get_CalenderAlarmValue ( rtc_alarm_value_t * const alarm_val );
```

[引数]

I/O	引数	説明
I	rtc_alarm_value_t * const *alarm_val*;	読み出した発生条件を格納する構造体のポインタ

備考　アラーム発生条件 rtc_alarm_value_t についての詳細は、R_RTC_Set_AlarmValue を参照してください。

[戻り値]

なし

R_RTC_Get_BinaryAlarmValue

アラームの発生条件を読み出します。（バイナリモード設定時）

[指定形式]

```
#include   “r_cg_macrodriver.h”
#include   “r_cg_rtc.h”
void       R_RTC_Get_BinaryAlarmValue ( uint32_t * const alarm_enable,
 uint32_t * const alarm_val );
```

[引数]

I/O	引数	説明
I	uint32_t * const *alarm_enable*;	読み出したアラームイネーブル値を格納する変数へのポインタ
I	uint32_t * const *alarm_val*;	読み出した発生条件を格納する変数へのポインタ

[戻り値]

なし

r_rtc_callback_alarm

アラーム割り込み INTRTC の発生に伴う処理を行います。

備考　本 API 関数は、アラーム割り込み INTRTC に対応した割り込み処理 r_rtc_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_rtc_callback_alarm ( void );
```

[引数]

なし

[戻り値]

なし

R_RTC_Set_RTC1HZOn

RTC1HZ 端子に対する補正クロック（1Hz）の出力を許可します。

[指定形式]

```
void        R_RTC_Set_RTC1HZOn ( void );
```

[引数]

なし

[戻り値]

なし

R_RTC_Set_RTC1HZOff

RTC1HZ 端子に対する補正クロック（1Hz）の出力を禁止します。

[指定形式]

```
void        R_RTC_Set_RTC1HZOff ( void );
```

[引数]

なし

[戻り値]

なし

R_RTC_Set_RTCOUTOn

RTCOUT の出力を許可します。

[指定形式]

```
void        R_RTC_Set_RTCOUTOn ( void );
```

[引数]

なし

[戻り値]

なし

R_RTC_Set_RTCOUTOff

RTCOUT の出力を禁止します。

[指定形式]

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

[引数]

なし

[戻り値]

なし

r_rtc_alarminterrupt

アラーム割り込み INTRTCALM の発生に伴う処理を行います。

備考　本 API 関数は、アラーム割り込み INTRTCALM 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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_rtc_alarminterrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_rtc_alarminterrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_rtc_periodinterrupt

周期割り込み INTRTCPRD の発生に伴う処理を行います。

備考　本 API 関数は、周期割り込み INTRTCPRD 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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_rtc_periodinterrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_rtc_periodinterrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_rtc_callback_periodic

定周期割り込み INTRTC の発生に伴う処理を行います。

備考　本 API 関数は、定周期割り込み INTRTC に対応した割り込み処理 r_rtc_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_rtc_callback_periodic ( void );
```

[引数]

なし

[戻り値]

なし

使用例

初回アラーム割り込みから同時間内は 10 分おきに割り込みを発生させる

[GUI 設定例]

リアルタイム・クロック				使用する
	RTC			使用する
			リアルタイムクロック動作設定	使用する
			時間制の選択	24 時間制
			リアルタイムクロック初期値設定	使用する 04/01/2018 00:00:00
			RTC1HZ 端子の出力(1 Hz)許可	使用しない
			アラーム検出機能	使用する
			アラーム検出初期値	使用する
			曜日	日曜日 木曜日 月曜日 金曜日 火曜日 土曜日 水曜日
			時 : 分	午前 07:00
			アラーム割り込み機能(INTRTC)	使用する
			定周期割り込み機能(INTRTC)	使用しない
			優先順位(INTRTC)	低

[API 設定例]

r_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Enable the real-time clock */
    R_RTC_Start();

    /* Start the alarm operation */
    R_RTC_Set_AlarmOn();

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

r_cg_rtc_user.c

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

static void r_rtc_callback_alarm(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Get alarm value */
    R_RTC_Get_AlarmValue((rtc_alarm_value_t *)&alarm_val);

    if ((alarm_val.alarmwm + 0x10U) <= 0x59U)
    {
        alarm_val.alarmwm += 0x10U;
        /* Set alarm value */
        R_RTC_Set_AlarmValue(alarm_val);
    }
    else
    {
        /* Stop the alarm operation */
        R_RTC_Set_AlarmOff();
    }
    /* End user code. Do not edit comment generated here */
}
```

3.3.14 サブシステム・クロック周波数測定回路

以下に、コード生成ツールがサブシステム・クロック周波数測定回路用として出力する API 関数の一覧を示します。

表 3.15 サブシステム・クロック周波数測定回路用 API 関数

API 関数名	機能概要
R_FMC_Create	サブシステム・クロック周波数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_FMC_Create_UserInit	サブシステム・クロック周波数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_fmc_interrupt	周波数測定完了割り込み INTFM の発生に伴う処理を行います。
R_FMC_Start	サブシステム・クロック周波数測定回路を利用した周波数の測定を開始します。
R_FMC_Stop	サブシステム・クロック周波数測定回路を利用した周波数の測定を終了します。
R_FMC_Set_PowerOff	サブシステム・クロック周波数測定回路に対するクロック供給を停止します。

R_FMC_Create

サブシステム・クロック周波数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。

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

[指定形式]

```
void    R_FMC_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_FMC_Create_UserInit

サブシステム・クロック周波数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_FMC_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_FMC_Create_UserInit ( void );
```

[引数]
なし

[戻り値]
なし

r_fmc_interrupt

周波数測定完了割り込み INTFM の発生に伴う処理を行います。

備考　本 API 関数は、周波数測定完了割り込み INTRM 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]

CA78K0R コンパイラの場合

```
__interrupt static void      r_fmc_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void      __near r_fmc_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_FMC_Start

サブシステム・クロック周波数測定回路を利用した周波数の測定を開始します。

[指定形式]

```
void        R_FMC_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_FMC_Stop

サブシステム・クロック周波数測定回路を利用した周波数の測定を終了します

[指定形式]

```
void        R_FMC_stop ( void );
```

[引数]

なし

[戻り値]

なし

R_FMC_Set_PowerOff

サブシステム・クロック周波数測定回路に対するクロック供給を停止します。

備考　本 API 関数の呼び出しにより、サブシステム・クロック周波数測定回路はリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

```
void    R_FMC_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

3.3.15 12 ビット・インターバル・タイマ

以下に、コード生成ツールが 12 ビット・インターバル・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 12 ビット・インターバル・タイマ用 API 関数

API 関数名	機能概要
R_IT_Create	12 ビット・インターバル・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_IT_Create_UserInit	12 ビット・インターバル・タイマに関するユーザ独自の初期化処理を行います。
r_it_interrupt	12 ビット・インターバル・タイマ割り込み INTIT の発生に伴う処理を行います。
R_IT_Start	12 ビット・インターバル・タイマのカウントを開始します。
R_IT_Stop	12 ビット・インターバル・タイマのカウントを終了します。
R_IT_Reset	12 ビット・インターバル・タイマをリセットします。
R_IT_Set_PowerOff	12 ビット・インターバル・タイマに対するクロック供給を停止します。

R_IT_Create

12 ビット・インターバル・タイマを制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_IT_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_IT_Create_UserInit

12 ビット・インターバル・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_IT_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_IT_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_it_interrupt

12 ビット・インターバル・タイマ割り込み INTIT の発生に伴う処理を行います。

備考　本 API 関数は、12 ビット・インターバル・タイマ割り込み INTIT 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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_it_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_it_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_IT_Start

12 ビット・インターバル・タイマのカウントを開始します。

備考　　カウント動作停止で、タイマはクリアされます。

[指定形式]

```
void    R_IT_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_IT_Stop

12 ビット・インターバル・タイマのカウントを終了します。

備考　　カウント動作停止で、タイマはクリアされます。

[指定形式]

```
void　　R_IT_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_IT_Reset

12 ビット・インターバル・タイマをリセットします。

[指定形式]

```
void        R_IT_Reset ( void );
```

[引数]

なし

[戻り値]

なし

R_IT_Set_PowerOff

12 ビット・インターバル・タイマに対するクロック供給を停止します。

備考 1. 本 API 関数の呼び出しにより、12 ビット・インターバル・タイマはリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

備考 2. 本 API 関数では、周辺イネーブル・レジスタ *n* の RTCEN ビットを操作することにより、12 ビット・インターバル・タイマに対するクロック供給の停止を実現しています。
このため、本 API 関数の呼び出しを行った際には、RTCEN ビットを共用しているほかの周辺装置（リアルタイム・クロックなど）に対するクロック供給も停止することになります。

[指定形式]

```
void        R_IT_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

使用例

ワンショットタイマとして使用する

[GUI 設定例]

12 ビット・インターバル・タイマ				使用する
	IT			使用する
			インターバル・タイマ動作設定	使用する
			インターバル時間	100ms (実際の値：100)
			インターバル信号検出(INTIT)	使用する
			優先順位	低

[API 設定例]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start IT module operation */
    R_IT_Start();

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

r_cg_it_user.c

```
static void __near r_it_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop IT module operation */
    R_IT_Stop();
    /* End user code. Do not edit comment generated here */
}
```

3.3.16 8 ビット・インターバル・タイマ

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

表 3.17 8 ビット・インターバル・タイマ用 API 関数

API 関数名	機能概要
R_IT8Bitm_Channeln_Create	8 ビット・インターバル・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_IT8Bitm_Channeln_Create_UserInit	8 ビット・インターバル・タイマに関するユーザ独自の初期化処理を行います。
r_it8bitm_channeln_interrupt	8 ビット・インターバル・タイマ割り込み INTIT*n*0、または INTIT*n*1 の発生に伴う処理を行います。
R_IT8Bitm_Channeln_Start	8 ビット・インターバル・タイマのカウントを開始します。
R_IT8Bitm_Channeln_Stop	8 ビット・インターバル・タイマのカウントを終了します。
R_IT8Bitm_Channeln_Set_PowerOff	8 ビット・インターバル・タイマに対するクロック供給を停止します。
R_IT8Bitm_Set_PowerOff	8 ビット・インターバル・タイマに対するクロック供給を停止します。

R_IT8Bit*m*_Channel*n*_Create

8 ビット・インターバル・タイマを制御するうえで必要となる初期化処理を行います。

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

[指定形式]

void	R_IT8Bit*m*_Channel*n*_Create (void);

備考　*m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_IT8Bit*m*_Channel*n*_Create_UserInit

8 ビット・インターバル・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_IT8Bitm_Channeln_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_IT8Bit*m*_Channel*n*_Create_UserInit (void);

備考　*m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

r_it8bit*m*_channel*n*_interrupt

8 ビット・インターバル・タイマ割り込み INTIT*n*0、または INTIT*n*1 の発生に伴う処理を行います。

備考　本 API 関数は、8 ビット・インターバル・タイマ割り込み INTIT*n*0、または INTIT*n*1 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]

CA78K0R コンパイラの場合

```
__interrupt static void    r_it8bitm_channeln_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void    __near r_it8bitm_channeln_interrupt ( void );
```

備考　*m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_IT8Bit*m*_Channel*n*_Start

8 ビット・インターバル・タイマのカウントを開始します。

[指定形式]

void	R_IT8Bit*m*_Channel*n*_Start (void);

備考 *m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_IT8Bit*m*_Channel*n*_Stop

8 ビット・インターバル・タイマのカウントを終了します。

[指定形式]

void	R_IT8Bit*m*_Channel*n*_Stop (void);

備考 *m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_IT8Bit*m*_Channel*n*_Set_PowerOff

8 ビット・インターバル・タイマに対するクロック供給を停止します。

備考　本 API 関数の呼び出しにより、8 ビット・インターバル・タイマはリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

void	R_IT8Bit*m*_Channel*n*_Set_PowerOff (void);

備考　*m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_IT8Bit*m*_Set_PowerOff

8 ビット・インターバル・タイマに対するクロック供給を停止します。

備考　本 API 関数の呼び出しにより、8 ビット・インターバル・タイマはリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

void	R_IT8Bit*m*_Set_PowerOff (void);

備考　*m* は、ユニット番号をを意味します。

[引数]

なし

[戻り値]

なし

使用例

ワンショットタイマとして使用する

[GUI 設定例]

8 ビット・インターバル・タイマ				使用する
	IT8bit0			使用する
		Channel 0		使用する
			チャネル 0	8 bit
			インターバル・タイマ動作設定	自動
			インターバル時間	100ms (実際の値：7.8125)
			コンペアマッチ検出 (INTIT00)	使用する
			優先順位	低

[API 設定例]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start 8 bit interval timer unit0 Channel0 operation */
    R_IT8Bit0_Channel0_Start();

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

r_cg_it8bit_user.c

```
static void __near r_it8bit0_channel0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop 8 bit interval timer unit0 Channel0 operation */
    R_IT8Bit0_Channel0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

3.3.17 16 ビット・ウェイクアップ・タイマ

以下に、コード生成ツールが 16 ビット・ウェイクアップ・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 16 ビット・ウェイクアップ・タイマ用 API 関数

API 関数名	機能概要
R_WUTM_Create	16 ビット・ウェイクアップ・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_WUTM_Create_UserInit	16 ビット・ウェイクアップ・タイマに関するユーザ独自の初期化処理を行います。
r_wutm_interrupt	タイマ割り込みの発生に伴う処理を行います。
R_WUTM_Start	16 ビット・ウェイクアップ・タイマのカウント処理を開始します。
R_WUTM_Stop	16 ビット・ウェイクアップ・タイマのカウント処理を終了します。
R_WUTM_Set_PowerOff	16 ビット・ウェイクアップ・タイマに対するクロック供給を停止します。

R_WUTM_Create

16 ビット・ウェイクアップ・タイマを制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_WUTM_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_WUTM_Create_UserInit

16 ビット・ウェイクアップ・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_WUTM_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_WUTM_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_wutm_interrupt

タイマ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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_wutm_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_wutm_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_WUTM_Start

16 ビット・ウェイクアップ・タイマのカウント処理を開始します。

[指定形式]

```
void        R_WUTM_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_WUTM_Stop

16 ビット・ウェイクアップ・タイマのカウント処理を終了します。

[指定形式]

```
void        R_WUTM_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_WUTM_Set_PowerOff

16 ビット・ウェイクアップ・タイマに対するクロック供給を停止します。

備考　本 API 関数の呼び出しにより、16 ビット・インターバル・タイマはリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

```
void    R_WUTM_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

使用例

ワンショットタイマとして使用する

[GUI 設定例]

タイマ				使用する
	TAU0			使用しない
	WUTM			使用する
			16 ビット・ウエイクアップ・タイマ動作設定	使用する
			カウント・クロック設定	自動
			インターバル時間	100ms (実際の値：100)
			設定した周期毎に通知する (INTWUTM)	使用する
			優先順位(INTWUTM)	低

[API 設定例]

r_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start WUTM counter */
    R_WUTM_Start();

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

r_cg_timer_user.c

```
static void __near r_wutm_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop WUTM counter */
    R_WUTM_Stop();
    /* End user code. Do not edit comment generated here */
}
```

3.3.18 クロック出力／ブザー出力制御回路

以下に、コード生成ツールがクロック出力／ブザー出力制御回路用として出力する API 関数の一覧を示します。

表 3.19 クロック出力／ブザー出力制御回路用 API 関数

API 関数名	機能概要
R_PCLBUZn_Create	クロック出力／ブザー出力制御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_PCLBUZn_Create_UserInit	クロック出力／ブザー出力制御回路に関するユーザ独自の初期化処理を行います。
R_PCLBUZn_Start	クロック出力／ブザー出力を開始します。
R_PCLBUZn_Stop	クロック出力／ブザー出力を終了します。
R_PCLBUZn_Set_PowerOff	クロック出力／ブザー出力制御回路に対するクロック供給を停止します。

R_PCLBUZ*n*_Create

クロック出力／ブザー出力制御回路を制御するうえで必要となる初期化処理を行います。

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

[指定形式]

void	R_PCLBUZ*n*_Create (void);

備考　*n* は、出力端子を意味します。

[引数]

なし

[戻り値]

なし

R_PCLBUZ*n*_Create_UserInit

クロック出力／ブザー出力制御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_PCLBUZn_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_PCLBUZ*n*_Create_UserInit (void);

備考　*n* は、出力端子を意味します。

[引数]

なし

[戻り値]

なし

R_PCLBUZ*n*_Start

クロック出力／ブザー出力を開始します。

[指定形式]

void	R_PCLBUZ*n*_Start (void);

備考 *n* は、出力端子を意味します。

[引数]

なし

[戻り値]

なし

R_PCLBUZ*n*_Stop

クロック出力／ブザー出力を終了します。

[指定形式]

void	R_PCLBUZ*n*_Stop (void);

備考 *n* は、出力端子を意味します。

[引数]

なし

[戻り値]

なし

R_PCLBUZ*n*_Set_PowerOff

クロック出力／ブザー出力制御回路に対するクロック供給を停止します。

備考 1. 本 API 関数の呼び出しにより、クロック出力／ブザー出力制御回路はリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

備考 2. 本 API 関数では、周辺イネーブル・レジスタ *n* の RTCEN ビットを操作することにより、クロック出力／ブザー出力制御回路に対するクロック供給の停止を実現しています。
このため、本 API 関数の呼び出しを行った際には、RTCEN ビットを共用しているほかの周辺装置（リアルタイム・クロックなど）に対するクロック供給も停止することになります。

[指定形式]

void	R_PCLBUZ*n*_Set_PowerOff (void);

備考 *n* は、出力端子を意味します。

[引数]

なし

[戻り値]

なし

使用例

出力を開始する

[GUI 設定例]

クロック出力/ブザー出力				使用する
	PCLBUZ0			使用する
			クロック出力／ブザー出力動作設定	使用する
			PCLBUZ0 の出力クロックの選択	1.875 (fSL/2^3)(kHz)
			スルー・レート	使用しない

[API 設定例]

r_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start the PCLBUZ0 module */
    R_PCLBUZ0_Start();

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

3.3.19 ウォッチドッグ・タイマ

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

表 3.20 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_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	インターバル・タイマ割り込み INTWDTI の発生に伴う処理を行います。
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

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。

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

[指定形式]

```
void    R_WDT_Create ( 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]

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

[引数]

なし

[戻り値]

なし

r_wdt_interrupt

インターバル・タイマ割り込み INTWDTI の発生に伴う処理を行います。

備考　本 API 関数は、インターバル・タイマ割り込み INTWDTI 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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_wdt_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_wdt_interrupt ( 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

使用例

ウォッチドッグ・タイマのインターバル割り込みにてフラグが立っていればカウンタクリアする

[GUI 設定例]

ウォッチドッグ・タイマ				使用する
	WDT			使用する
			ウォッチドッグ・タイマ動作設定	使用する
			HALT/STOP/SNOOZE モード時の動作設定	許可
			オーバフロー時間	136.53 (2^11/fWDT)(ms)
			ウインドウ・オープン期間	100(%)
			オーバフロー時間の 75% + 1/2fIL 到達時にインターバル割り込みを発生する(INTWDTI)	使用する
			優先順位	低

[API 設定例]

r_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    while (1U)
    {
        /* Restart the watchdog timer */
        R_WDT_Restart();
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_wdt_user.c

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

static void __near r_wdt_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    if(g_wdt_f == 1U)
    {
        /* Restart the watchdog timer */
        R_WDT_Restart();
    }
    /* End user code. Do not edit comment generated here */
}
```

3.3.20 プログラマブル・ゲイン計装アンプ付き 24 ビット ΔΣA/D コンバータ

以下に、コード生成ツールがプログラマブル・ゲイン計装アンプ付き 24 ビット ΔΣA/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 3.21 プログラマブル・ゲイン計装アンプ付き 24 ビット ΔΣA/D コンバータ用 API 関数

API 関数名	機能概要
R_PGA_DSAD_Create	プログラマブル・ゲイン計装アンプ付き 24 ビット ΔΣA/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_PGA_DSAD_Create_UserInit	プログラマブル・ゲイン計装アンプ付き 24 ビット ΔΣA/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_pga_dsad_interrupt_conversion	24 ビット ΔΣA/D コンバータ変換終了割り込み INTDSAD の発生に伴う処理を行います。
r_pga_dsad_interrupt_scan	24 ビット ΔΣA/D コンバータスキャン完了割り込み INTDSADS の発生に伴う処理を行います。
R_PGA_DSAD_Start	A/D 変換を開始します。
R_PGA_DSAD_Stop	A/D 変換を終了します。
R_PGA_DSAD_Set_PowerOff	プログラマブル・ゲイン計装アンプ付き 24 ビット ΔΣA/D コンバータに対するクロックの供給を停止します。
R_PGA_DSAD_Get_AverageResult	A/D 変換結果の平均値を読み出します。
R_PGA_DSAD_Get_Result	A/D 変換結果を読み出します。
R_PGA_DSAD_CAMP_OffsetTrimming	コンフィギュラブル・アンプをプログラマブル・ゲイン計装アンプ付き 24 ビット ΔΣA/D コンバータへ接続し、オフセット・トリミングを行います。
r_pga_dsad_conversion _interrupt	24 ビット ΔΣA/D コンバータ変換終了割り込み INTDSAD の発生に伴う処理を行います。
r_pga_dsad_scan _interrupt	24 ビット ΔΣA/D コンバータスキャン完了割り込み INTDSADS の発生に伴う処理を行います。

R_PGA_DSAD_Create

プログラマブル・ゲイン計装アンプ付き 24 ビット ΔΣA/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。

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

[指定形式]

```
void    R_PGA_DSAD_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_PGA_DSAD_Create_UserInit

プログラマブル・ゲイン計装アンプ付き 24 ビット ΔΣA/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_PGA_DSAD_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_PGA_DSAD_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_pga_dsad_interrupt_conversion

24 ビット ΔΣA/D コンバータ変換終了割り込み INTDSAD の発生に伴う処理を行います。

備考　本 API 関数は、24 ビット ΔΣA/D コンバータ変換終了割り込み INTDSAD 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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_pga_dsad_interrupt_conversion ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_pga_dsad_interrupt_conversion ( void );
```

[引数]

なし

[戻り値]

なし

r_pga_dsad_interrupt_scan

24 ビット ΔΣA/D コンバータスキャン完了割り込み INTDSADS の発生に伴う処理を行います。

備考　本 API 関数は、24 ビット ΔΣA/D コンバータスキャン完了割り込み INTDSADS 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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_pga_dsad_interrupt_scan ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_pga_dsad_interrupt_scan ( void );
```

[引数]

なし

[戻り値]

なし

R_PGA_DSAD_Start

A/D 変換を開始します。

[指定形式]

```
void        R_PGA_DSAD_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_PGA_DSAD_Stop

A/D 変換を終了します。

[指定形式]

```
void        R_PGA_DSAD_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_PGA_DSAD_Set_PowerOff

プログラマブル・ゲイン計装アンプ付き 24 ビット ΔΣA/D コンバータに対するクロック供給を停止します。

[指定形式]

```
void        R_PGA_DSAD_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

R_PGA_DSAD_Get_AverageResult

A/D 変換結果の平均値を読み出します。

[指定形式]

```
#include   “r_cg_macrodriver.h”
void       R_PGA_DSAD_Get_AverageResult ( uint16_t * const bufferH, uint16_t * const bufferL );
```

[引数]

I/O	引数	説明
O	uint16_t * const *bufferH*;	読み出した A/D 変換結果（DSADMVM レジスタと DSADMVH レジスタ）を格納する領域へのポインタ
O	uint16_t * const *bufferLl*;	読み出した A/D 変換結果（DSADMVC レジスタと DSADMVL レジスタ）を格納する領域へのポインタ

[戻り値]

なし

R_PGA_DSAD_Get_Result

A/D 変換結果を読み出します。

[指定形式]

```
#include   “r_cg_macrodriver.h”
void       R_PGA_DSAD_Get_Result ( uint16_t * const bufferH, uint16_t * const bufferL );
```

[引数]

I/O	引数	説明
O	uint16_t * const *bufferH*;	読み出した A/D 変換結果（DSADMVM レジスタと DSADMVH レジスタ）を格納する領域へのポインタ
O	uint16_t * const *bufferLl*;	読み出した A/D 変換結果（DSADMVC レジスタと DSADMVL レジスタ）を格納する領域へのポインタ

[戻り値]

なし

R_PGA_DSAD_CAMP_OffsetTrimming

コンフィギュラブル・アンプをプログラマブル・ゲイン計装アンプ付き 24 ビット ΔΣA/D コンバータへ接続し、オフセット・トリミングを行います。

[指定形式]

```
void        R_PGA_DSAD_CAMP_OffsetTrimming ( void );
```

[引数]

なし

[戻り値]

なし

r_pga_dsad_conversion _interrupt

24 ビット ΔΣA/D コンバータ変換終了割り込み INTDSAD の発生に伴う処理を行います。

備考　本 API 関数は、24 ビット ΔΣA/D コンバータ変換終了割り込み INTDSAD 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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_pga_dsad_conversion _interrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_pga_dsad_conversion _interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_pga_dsad_scan _interrupt

24 ビット ΔΣA/D コンバータスキャン完了割り込み INTDSADS の発生に伴う処理を行います。

備考　本 API 関数は、24 ビット ΔΣA/D コンバータスキャン完了割り込み INTDSADS 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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_pga_dsad_scan _interrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_pga_dsad_scan _interrupt ( void );
```

[引数]

なし

[戻り値]

なし

3.3.21 A/D コンバータ

以下に，コード生成ツールが 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 3.22 A/D コンバータ用 API 関数

API 関数名	機能概要
R_ADC_Create	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_ADC_Create_UserInit	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_adc_interrupt	A/D 変換終了割り込み INTAD の発生に伴う処理を行います。
R_ADC_Set_OperationOn	電圧コンパレータを動作許可状態に設定します。
R_ADC_Set_OperationOff	電圧コンパレータを動作停止状態に設定します。
R_ADC_Start	A/D 変換を開始します。
R_ADC_Stop	A/D 変換を終了します。
R_ADC_Reset	A/D コンバータをリセットします。
R_ADC_Set_PowerOff	A/D コンバータに対するクロック供給を停止します。
R_ADC_Set_ADChannel	A/D 変換するアナログ電圧の入力端子を設定します。
R_ADC_Set_SnoozeOn	STOP モードから SNOOZE モードへの切り替えを許可します。
R_ADC_Set_SnoozeOff	STOP モードから SNOOZE モードへの切り替えを禁止します。
R_ADC_Set_TestChannel	A/D コンバータの動作モードを設定します。
R_ADC_Get_Result	変換結果を獲得します。
R_ADC_Get_Result_8bit	A/D 変換結果（10 ビット）を読み出します。
r_s12adn_compare_interrupt	A/D 変換結果（8 ビット：10 ビット分解能の上位 8 ビット）を読み出します。

R_ADC_Create

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。

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

[指定形式]

```
void    R_ADC_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_ADC_Create_UserInit

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_ADC_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_ADC_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_adc_interrupt

A/D 変換終了割り込み INTAD の発生に伴う処理を行います。

備考　本 API 関数は，A/D 変換終了割り込み INTAD 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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_adc_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_adc_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_ADC_Set_OperationOn

電圧コンパレータを動作許可状態に設定します。

備考 1. 電圧コンパレータが動作停止状態から動作許可状態へと移行した際、約 1μ 秒の安定時間を必要とします。
したがって、本 API 関数と R_ADC_Start の間には、約 1μ 秒の時間を空ける必要があります。

備考 2. [A/D コンバータ]の[コンパレータ動作設定]エリアで“許可”を選択した場合、電圧コンパレータは“常時 ON”となるため、本 API 関数の呼び出しは不要となります。

[指定形式]

```
void        R_ADC_Set_OperationOn ( void );
```

[引数]

なし

[戻り値]

なし

R_ADC_Set_OperationOff

電圧コンパレータを動作禁止状態に設定します。

[指定形式]

```
void        R_ADC_Set_OperationOff ( void );
```

[引数]

なし

[戻り値]

なし

R_ADC_Start

A/D 変換を開始します。

備考 電圧コンパレータが動作停止状態から動作許可状態へと移行した際、約 1μ 秒の安定時間を必要とします。
したがって、R_ADC_Set_OperationOn と本 API 関数の間には、約 1μ 秒の時間を空ける必要があります。

[指定形式]

void	R_S12AD*n*_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_ADC_Stop

A/D 変換を終了します。

備考 電圧コンパレータは、本 API 関数の処理完了後も動作を継続しています。
したがって、電圧コンパレータの動作を停止する場合は、本 API 関数の処理完了後、R_ADC_Set_OperationOff を呼び出す必要があります。

[指定形式]

```
void    R_ADC_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_ADC_Reset

A/D コンバータをリセットします。

[指定形式]

```
void        R_ADC_Reset ( void );
```

[引数]

なし

[戻り値]

なし

R_ADC_Set_PowerOff

A/D コンバータに対するクロック供給を停止します。

備考　本 API 関数の呼び出しにより、A/D コンバータはリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

```
void        R_ADC_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

R_ADC_Set_ADChannel

A/D 変換するアナログ電圧の入力端子を設定します。

備考　　引数 *channel* に指定された値は、アナログ入力チャネル指定レジスタ（ADS）に設定されます。

[指定形式]

```
#include      “r_cg_macrodriver.h”
#include      “r_cg_adc.h”
MD_STATUS   R_ADC_Set_ADChannel ( ad_channel_t channel );
```

[引数]

I/O	引数	説明
I	ad_channel_t *channel*;	アナログ電圧の入力端子 ADCHANNEL*n*：　　入力端子

[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_ADC_Set_SnoozeOn

STOP モードから SNOOZE モードへの切り替えを許可します。

[指定形式]

```
void        R_ADC_Set_SnoozeOn ( void );
```

[引数]

なし

[戻り値]

なし

R_ADC_Set_SnoozeOff

STOP モードから SNOOZE モードへの切り替えを禁止します。

[指定形式]

```
void        R_ADC_Set_SnoozeOff ( void );
```

[引数]

なし

[戻り値]

なし

R_ADC_Set_TestChannel

A/D コンバータの動作モードを設定します。

[指定形式]

```
#include        “r_cg_macrodriver.h”
#include        “r_cg_adc.h”
MD_STATUS   R_ADC_Set_TestChannel ( test_channel_t channel );
```

[引数]

I/O	引数	説明
I	test_channel_t *channel*;	A/D コンバータの動作モード ADNORMALINPUT： 通常モード（通常の A/D 変換） ADNORMALINPUT： テストモード（AVREFM 入力電圧） ADNORMALINPUT： テストモード（AVREFP 入力電圧）

[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_ADC_Get_Result

A/D 変換結果（10 ビット）を読み出します。

[指定形式]

```
#include   “r_cg_macrodriver.h”
void       R_ADC_Get_Result ( uint16_t * const buffer );
```

[引数]

I/O	引数	説明
O	uint16_t * const *buffer*;	読み出した A/D 変換結果を格納する領域へのポインタ

[戻り値]

なし

R_ADC_Get_Result_8bit

A/D 変換結果（8 ビット：10 ビット分解能の上位 8 ビット）を読み出します。

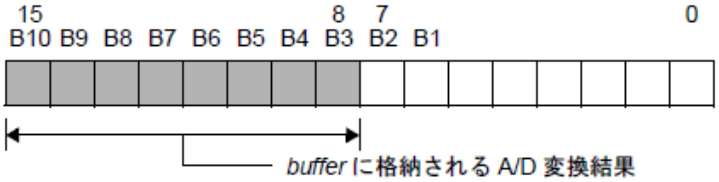
[指定形式]

```
#include    “r_cg_macrodriver.h”
void        R_ADC_Get_Result ( uint8_t * const buffer);
```

[引数]

I/O	引数	説明
O	Uint8_t * const *buffer*;	読み出した A/D 変換結果を格納する領域へのポインタ

備考　以下に、*buffer* に格納される A/D 変換結果を示します。



[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

使用例

2 端子の AD 変換結果を順に取得する

[GUI 設定例]

A/D コンバータ				使用する
	ADC			使用する
			A/D コンバータ動作設定	使用する
			コンパレータ動作設定	許可
			分解能設定	8 ビット
			VREF(+)設定	VDD
			VREF(-)設定	VSS
			トリガ・モード設定	ソフトウエア・トリガ・モード
			動作モード設定	ワンショット・セレクト・モード
			ANI0 - ANI23 アナログ入力端子設定	ANI0 - ANI1
			変換開始チャネル設定	ANI0
			変換時間モード	標準 1
			変換時間	34 (1088/fCLK)(μs)
			変換結果上限/下限値設定	ADLL≦ADCRH≦ADUL で割り込み要求信号(INTAD)を発生
			上限値(ADUL)	255
			下限値(ADLL)	0
			A/D の割り込み許可(INTAD)	使用する
			優先順位	低

[API 設定例]

r_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start the AD converter */
    R_ADC_Start();

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

r_cg_adc_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_adc_ch000_value;
volatile uint8_t g_adc_ch001_value;
/* End user code. Do not edit comment generated here */

static void __near r_adc_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop the AD converter */
    R_ADC_Stop();

    if(ADS == (uint8_t)ADCHANNEL0)
    {
        /* Return the higher 8 bits conversion result */
        R_ADC_Get_Result_8bit((uint8_t *)&g_adc_ch000_value);

        /* Start the AD converter */
        R_ADC_Set_ADChannel(ADCHANNEL1);
        R_ADC_Start();
    }
    else
    {
        /* Return the higher 8 bits conversion result */
        R_ADC_Get_Result_8bit((uint8_t *)&g_adc_ch001_value);
    }
    /* End user code. Do not edit comment generated here */
}
```

3.3.22 コンフィギュラブル・アンプ

以下に，コード生成ツールがコンフィギュラブル・アンプ用として出力する API 関数の一覧を示します。

表 3.23 コンフィギュラブル・アンプ用 API 関数

API 関数名	機能概要
R_CAMP_Create	コンフィギュラブル・アンプを制御するうえで必要となる初期化処理を行います。
R_CAMP_Create_UserInit	コンフィギュラブル・アンプに関するユーザ独自の初期化処理を行います。
R_CAMPn_Start	コンフィギュラブル・アンプ *n*（AMP*n*）の電源をオンにします。
R_CAMPn_Stop	コンフィギュラブル・アンプ *n*（AMP*n*）の電源をオフにします。
R_CAMP_Set_PowerOff	コンフィギュラブル・アンプに対するクロック供給を停止します。

R_CAMP_Create

コンフィギュラブル・アンプを制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_CAMP_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_CAMP_Create_UserInit

コンフィギュラブル・アンプ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_CAMP_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_CAMP_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

R_CAMP*n*_Start

コンフィギュラブル・アンプ *n*（AMP*n*）の電源をオンにします。

[指定形式]

void	R_CAMP*n*_Start (void);

備考 *n* は、チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_CAMP*n*_Stop

コンフィギュラブル・アンプ *n*（AMP*n*）の電源をオフにします。

[指定形式]

void	R_CAMP*n*_Stop (void);

備考 *n* は、チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_CAMP_Set_PowerOff

コンフィギュラブル・アンプに対するクロック供給を停止します。

[指定形式]

```
void        R_CAMP_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

3.3.23 温度センサ

以下に，コード生成ツールが温度センサ用として出力する API 関数の一覧を示します。

表 3.24 温度センサ用 API 関数

API 関数名	機能概要
R_TMPS_Create	温度センサを制御するうえで必要となる初期化処理を行います。
R_TMPS_Create_UserInit	温度センサに関するユーザ独自の初期化処理を行います。
R_TMPS_Start	温度センサを利用した温度の計測を開始します。
R_TMPS_Stop	温度センサを利用した温度の計測を終了します。
R_TMPS_Reset	温度センサをリセットします。
R_TMPS_Set_PowerOff	温度センサに対するクロック供給を停止します。

R_TMPS_Create

温度センサを制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_TMPS_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_TMPS_Create_UserInit

温度センサ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_TMPS_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_TMPS_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

R_TMPS_Start

温度センサを利用した温度の計測を開始します。

[指定形式]

```
void        R_TMPS_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_TMPS_Stop

温度センサを利用した温度の計測を終了します。

[指定形式]

```
void        R_TMPS_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_TMPS_Reset

温度センサをリセットします。

[指定形式]

```
void        R_TMPS_Reset ( void );
```

[引数]

なし

[戻り値]

なし

R_TMPS_Set_PowerOff

温度センサに対するクロック供給を停止します。

備考　本 API 関数の呼び出しにより、温度センサはリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

```
void    R_TMPS_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

使用例

温度センサの出力電圧を A/D コンバータで測定することで温度を測定する

[GUI 設定例]

温度センサ				使用する
	TS			使用する
			温度センサ動作設定	使用する
			動作モード設定	常温域（mode 2）

A/D コンバータ				使用する
	ADC			使用する
			A/D コンバータ動作設定	使用する
			コンパレータ動作設定	停止
			分解能設定	8 ビット
			VREF(+)設定	内部基準電圧
			VREF(-)設定	VSS
			トリガ・モード設定	ソフトウエア・トリガ・モード
			動作モード設定	ワンショット・セレクト・モード
			ANI0 - ANI5 アナログ入力端子設定	すべて デジタル
			変換開始チャネル設定	温度センサ出力
			変換時間モード	標準 1
			変換時間	544/fCLK 22.6667(μs)
			変換結果上限/下限値設定	ADLL≦ADCRH≦ADUL で割り込み要求信号(INTAD)を発生
			上限値(ADUL)	255
			下限値(ADLL)	0
			A/D の割り込み許可(INTAD)	使用する
			優先順位	低

[API 設定例]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start the temperature sensor operation */
    R_TMPS_Start();

    /* Start the AD converter */
    R_ADC_Start();

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

r_cg_adc_user.c

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

static void __near r_adc_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop the AD converter */
    R_ADC_Stop();

    /* Return the higher 8 bits conversion result */
    R_ADC_Get_Result_8bit((uint8_t *)&g_adc_value);
    /* End user code. Do not edit comment generated here */
}
```

3.3.24 24 ビット ΔΣA/D コンバータ

以下に、コード生成ツールが 24 ビット ΔΣA/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 3.25 プログラマブル・ゲイン計装アンプ付き 24 ビット ΔΣA/D コンバータ用 API 関数

API 関数名	機能概要
R_DSADC_Create	24 ビット ΔΣA/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_DSADC_Create_UserInit	24 ビット ΔΣA/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_dsadc_interrupt	ΔΣA/D 変換終了割り込み INTDSAD の発生に伴う処理を行います。
r_dsadzcn_interrupt	ゼロクロス検出割り込み INTDSADZC*n* の発生に伴う処理を行います。
R_DSADC_Set_OperationOn	24 ビット ΔΣA/D コンバータを動作許可状態に設定します。
R_DSADC_Set_OperationOff	24 ビット ΔΣA/D コンバータを動作停止状態に設定します。
R_DSADC_Start	A/D 変換を開始します。
R_DSADC_Stop	A/D 変換を終了します。
R_DSADC_Reset	24 ビット ΔΣA/D コンバータをリセットします。
R_DSADC_Set_PowerOff	24 ビット ΔΣA/D コンバータに対する電荷リセットを行います。
R_DSADC_Channeln_Get_Result	A/D 変換結果（24 ビット）を読み出します。
R_DSADC_Channeln_Get_Result_16bit	A/D 変換結果（16 ビット：24 ビット分解能の上位 16 ビット）を読み出します。

R_DSADC_Create

24 ビット ΔΣA/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。

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

[指定形式]

```
void    R_DSADC_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_DSADC_Create_UserInit

24 ビット ΔΣA/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_DSADC_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_DSADC_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_dsadc_interrupt

ΔΣA/D 変換終了割り込み INTDSAD の発生に伴う処理を行います。

備考　本 API 関数は、ΔΣA/D 変換終了割り込み INTDSAD 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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_dsadc_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_dsadc_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_dsadzcn_interrupt

ゼロクロス検出割り込み INTDSADZCn の発生に伴う処理を行います。

備考　本 API 関数は、ゼロクロス検出割り込み INTDSADSZCn 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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_dsadzcn_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_dsadzcn_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_DSADC_Set_OperationOn

24 ビット ΔΣA/D コンバータを動作許可状態に設定します。

[指定形式]

```
void        R_DSADC_Set_OperationOn ( void );
```

[引数]

なし

[戻り値]

なし

R_DSADC_Set_OperationOff

24 ビット ΔΣA/D コンバータを動作禁止状態に設定します。

[指定形式]

```
void        R_DSADC_Set_OperationOff ( void );
```

[引数]

なし

[戻り値]

なし

R_DSADC_Start

A/D 変換を開始します。

[指定形式]

```
void        R_DSADC_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_DSADC_Stop

A/D 変換を終了します。

[指定形式]

```
void        R_DSADC_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_DSADC_Reset

24 ビット ΔΣA/D コンバータをリセットします。

[指定形式]

```
void        R_DSADC_Set_Reset ( void );
```

[引数]

なし

[戻り値]

なし

R_DSADC_Set_PowerOff

24 ビット ΔΣA/D コンバータに対する電荷リセットを行います。

備考　24 ビット ΔΣA/D コンバータに対す電荷リセットを行った場合、約 1μ 秒の安定時間を必要とします。

[指定形式]

```
void    R_DSADC_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

R_DSADC_Channel*n*_Get_Result

A/D 変換結果（24 ビット）の平均値を読み出します。

備考　本 API 関数による A/D 変換結果（24 ビット）の読み出しは、ΔΣA/D 変換終了割り込み INTDSAD の発生から ΔΣA/D 変換結果レジスタ *n* の最大保留時間内に行う必要があります。

[指定形式]

```
#include  “r_cg_macrodriver.h”
void      R_DSADC_Channeln_Get_Result ( uint32_t * const buffer );
```

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 *buffer*;	読み出した A/D 変換結果を格納する領域へのポインタ

[戻り値]

なし

R_DSADC_Channel*n*_Get_Result_16bit

A/D 変換結果（16 ビット：24 ビット分解能の上位 16 ビット）を読み出します。

備考　本 API 関数による A/D 変換結果の読み出しは、ΔΣA/D 変換終了割り込み INTDSAD の発生から ΔΣA/D 変換結果レジスタ *n* の最大保留時間内に行う必要があります。

[指定形式]

```
#include  “r_cg_macrodriver.h”
void      R_DSADC_Channeln_Get_Result_16bit ( uint16_t * const buffer );
```

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	Uint16_t * const *buffer*;	読み出した A/D 変換結果を格納する領域へのポインタ

[戻り値]

なし

3.3.25 D/A コンバータ

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

表 3.26 D/A コンバータ用 API 関数

API 関数名	機能概要
R_DAC_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_DAC_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_DACn_Start	D/A 変換を開始します。
R_DACn_Stop	D/A 変換を終了します。
R_DAC_Set_PowerOff	D/A コンバータに対するクロック供給を停止します。
R_DACn_Set_ConversionValue	ANO*n* 端子に出力するアナログ電圧値を設定します。
R_DACn_Change_OutputVoltage_8bit	D/A コンバータの出力電圧を変更します。（8 ビットモード）
R_DACn_Change_OutputVoltage	D/A コンバータの出力電圧を変更します。（12 ビットモード）
R_DACn_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_DAC_Reset	D/A コンバータをリセットします。

R_DAC_Create

D/A コンバータを制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_DAC_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_DAC_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_DAC_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_DAC_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

R_DAC*n*_Start

D/A 変換を開始します。

[指定形式]

void	R_DAC*n*_Start (void);

備考 *n* は、チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_DAC*n*_Stop

D/A 変換を終了します。

[指定形式]

void	R_DAC*n*_Stop (void);

備考 *n* は、チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_DAC_Set_PowerOff

D/A コンバータに対するクロック供給を停止します。

備考　本 API 関数の呼び出しにより、D/A コンバータはリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

```
void    R_DAC_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

R_DAC*n*_Set_ConversionValue

ANO*n* 端子に出力するアナログ電圧値を設定します。

[指定形式]

```
#include   “r_cg_macrodriver.h”
void       R_DAC*n*_Set_ConversionValue ( uint8_t *reg_value* );
```

備考　　*n* は、チャネル番号を意味します。

[引数]

I/O	引数	説明
I	Uint8_t *reg_value*;	D/A 変換を行うデータ

[戻り値]

なし

R_DAC*n*_Change_OutputVoltage_8bit

D/A コンバータの出力電圧を変更します。（8 ビットモード）

[指定形式]

```
#include   “r_cg_macrodriver.h”
void       R_DAC n_Change_OutputVoltage_8bit ( uint8_t outputVoltage );
```

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 *outputVoltage*;	出力電圧（下位 8 ビット）

[戻り値]

なし

R_DAC*n*_Change_OutputVoltage

D/A コンバータの出力電圧を変更します。（12 ビットモード）

[指定形式]

#include	“r_cg_macrodriver.h”
void	R_DAC*n*_Change_OutputVoltage (uint16_t *outputVoltage*);

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 *outputVoltage*;	出力電圧（下位 12 ビット）

[戻り値]

なし

R_DAC*n*_Create

D/A コンバータを制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
#include   “r_cg_macrodriver.h”
void       R_DAC*n*_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_DAC_Reset

D/A コンバータをリセットします。

[指定形式]

```
void        R_DAC_Reset ( void );
```

[引数]

なし

[戻り値]

なし

使用例

0x00 から開始した変換出力を一定周期毎に 0x10 ずつ上げて行き、0xFF になると変換停止

[GUI 設定例]

D/A コンバータ				使用する
	DAC			使用する
		DAC0		
			D/A コンバータ動作モード設定	使用する
			D/A コンバータ動作設定	通常モード
			アナログ出力 (ANO0)	使用する
			変換値	0x00

タイマ				使用する
	TAU0			使用する
		Channel 0		
			チャネル 0	インターバル・タイマ
			インターバル時間(16 ビット)	100ms (実際の値：100)
			カウント開始時に INTTM00 割り込みを発生する	使用しない
			タイマ・チャネル 0 のカウント完了で割り込み発生(INTTM00)	使用する
			優先順位 (INTTM00)	低

[API 設定例]

r_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start TAU0 channel 0 counter */
    R_TAU0_Channel0_Start();

    /* Enable the DA converter channel 0 */
    R_DAC0_Start();

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

r_cg_timer_user.c

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

/* Start user code for global. Do not edit comment generated here */
volatile uint16_t g_dac0_value = _00_DA0_CONVERSION_VALUE;
/* End user code. Do not edit comment generated here */

static void __near r_tau0_channel0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    g_dac0_value += 0x0010U;
    if (g_dac0_value <= 0x00FFU)
    {
        /* Set the DA converter channel 0 value */
        R_DAC0_Set_ConversionValue((uint8_t)g_dac0_value);
    }
    else
    {
        /* Stop the DA converter channel 0 */
        R_DAC0_Stop();

        /* Stop TAU0 channel 0 counter */
        R_TAU0_Channel0_Stop();
    }
    /* End user code. Do not edit comment generated here */
}
```

3.3.26 プログラマブル・ゲイン・アンプ

以下に，コード生成ツールがプログラマブル・ゲイン・アンプ用として出力する API 関数の一覧を示します。

表 3.27 プログラマブル・ゲイン・アンプ用 API 関数

API 関数名	機能概要
R_PGA_Create	プログラマブル・ゲイン・アンプを制御するうえで必要となる初期化処理を行います。
R_PGA_Create_UserInit	プログラマブル・ゲイン・アンプに関するユーザ独自の初期化処理を行います。
R_PGA_Start	プログラマブル・ゲイン・アンプの動作を開始します。
R_PGA_Stop	プログラマブル・ゲイン・アンプの動作を停止します。
R_PGA_Reset	プログラマブル・ゲイン・アンプをリセットします。
R_PGA_Set_PowerOff	プログラマブル・ゲイン・アンプに対するクロック供給を停止します。

R_PGA_Create

プログラマブル・ゲイン・アンプを制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_PGA_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_PGA_Create_UserInit

プログラマブル・ゲイン・アンプ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_PGA_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_PGA_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

R_PGA_Start

プログラマブル・ゲイン・アンプの動作を開始します。

[指定形式]

```
void        R_PGA_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_PGA_Stop

プログラマブル・ゲイン・アンプの動作を停止します。

[指定形式]

```
void        R_PGA_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_PGA_Reset

プログラマブル・ゲイン・アンプをリセットします。

[指定形式]

```
void        R_PGA_Reset ( void );
```

[引数]

なし

[戻り値]

なし

R_PGA_Set_PowerOff

プログラマブル・ゲイン・アンプに対するクロック供給を停止します。

備考　本 API 関数の呼び出しにより、プログラマブル・ゲイン・アンプはリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

```
void    R_PGA_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

3.3.27 コンパレータ

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

表 3.28 コンパレータ用 API 関数

API 関数名	機能概要
R_COMP_Create	コンパレータを制御するうえで必要となる初期化処理を行います。
R_COMP_Create_UserInit	コンパレータに関するユーザ独自の初期化処理を行います。
r_compn_interrupt	コンパレータ割り込み INTCMP*n* の発生に伴う処理を行います。
R_COMPn_Start	リファレンス入力電圧とアナログ入力電圧の比較動作を開始します。
R_COMPn_Stop	リファレンス入力電圧とアナログ入力電圧の比較動作を終了します。
R_COMP_Reset	コンパレータをリセットします。
R_COMP_Set_PowerOff	コンパレータに対するクロック供給を停止します。

R_COMP_Create

コンパレータを制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_COMP_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_COMP_Create_UserInit

コンパレータに関するユーザ独自の初期化処理を行います。

備考　　本 API 関数は，R_COMP_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_COMP_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_comp*n*_interrupt

コンパレータ割り込み INTCMP*n* の発生に伴う処理を行います。

備考　本 API 関数は、コンパレータ割り込み INTCMP*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]

CA78K0R コンパイラの場合

```
__interrupt static void    r_compn_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void    __near r_compn_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_COMP*n*_Start

リファレンス入力電圧とアナログ入力電圧の比較動作を開始します。

[指定形式]

void	R_COMP*n*_Start (void);

備考 *n* は、チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_COMP*n*_Stop

リファレンス入力電圧とアナログ入力電圧の比較動作を停止します。

[指定形式]

void	R_COMP*n*_Stop (void);

備考 *n* は、チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_COMP_Reset

コンパレータをリセットします。

[指定形式]

```
void        R_COMP_Reset ( void );
```

[引数]

なし

[戻り値]

なし

R_COMP_Set_PowerOff

コンパレータに対するクロック供給を停止します。

備考　本 API 関数の呼び出しにより、コンパレータはリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

```
void    R_COMP_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

使用例

比較結果の有効エッジを検出すると、比較終了

[GUI 設定例]

コンパレータ				使用する
	Comparator			使用する
			コンパレータ入力	IVCMP00
			基準電圧	IVREF0
			ノイズ・フィルタ使用許可	使用しない
			出力設定 (VCOUT0)	使用する
			出力極性設定	正転
			STOP モード解除 設定	使用する
			コンパレータ出力の有効エッジ検出で割り込み発生 (INTCMP0)	使用する
			有効エッジ	両エッジ
			優先順位	低

[API 設定例]

r_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start the comparator 0 */
    R_COMP0_Start();

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

r_cg_comp_user.c

```
static void __near r_comp0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop the comparator 0 */
    R_COMP0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

3.3.28 コンパレータ／プログラマブル・ゲイン・アンプ

以下に, コード生成ツールがコンパレータ／プログラマブル･ゲイン･アンプ用として出力する API 関数の一覧を示します。

表 3.29 コンパレータ／プログラマブル・ゲイン・アンプ用 API 関数

API 関数名	機能概要
R_COMPPGA_Create	コンパレータ／プログラマブル・ゲイン・アンプを制御するうえで必要となる初期化処理を行います。
R_COMPPGA_Set_PowerOff	コンパレータ／プログラマブル･ゲイン･アンプに対するクロック供給を停止します。
R_COMPPGA_Create_UserInit	コンパレータ／プログラマブル・ゲイン・アンプに関するユーザ独自の初期化処理を行います。
r_compn_interrupt	コンパレータ割り込み INTCMP*n* の発生に伴う処理を行います。
R_COMPn_Start	リファレンス入力電圧とアナログ入力電圧の比較動作を開始します。
R_COMPn_Stop	リファレンス入力電圧とアナログ入力電圧の比較動作を終了します。
R_PGA_Start	プログラマブル・ゲイン・アンプの動作を開始します。
R_PGA_Stop	プログラマブル・ゲイン・アンプの動作を停止します。
R_PWMOPT_Start	6 相 PWM オプション・ユニットに対する入力クロックを供給します。また、6 相 PWM オプション・ユニットの動作モードを設定します。
R_PWMOPT_Stop	6 相 PWM オプション・ユニットに対する入力クロックを停止します。

R_COMPPGA_Create

コンパレータ／プログラマブル・ゲイン・アンプを制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_COMPPGA_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_COMPPGA_Set_PowerOff

コンパレータ／プログラマブル・ゲイン・アンプに対するクロック供給を停止します。

備考　本 API 関数の呼び出しにより、コンパレータ／プログラマブル・ゲイン・アンプはリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

```
void    R_COMPPGA_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

R_COMPPGA_Create_UserInit

コンパレータ／プログラマブル・ゲイン・アンプ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_COMPPGA_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_COMPPGA_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_comp*n*_interrupt

コンパレータ割り込み INTCMP*n* の発生に伴う処理を行います。

備考　本 API 関数は，コンパレータ割り込み INTCMP*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]

CA78K0R コンパイラの場合

```
__interrupt static void    r_compn_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void    __near r_compn_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_COMP*n*_Start

リファレンス入力電圧とアナログ入力電圧の比較動作を開始します。

[指定形式]

void	R_COMP*n*_Start (void);

備考 *n* は、チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_COMP*n*_Stop

リファレンス入力電圧とアナログ入力電圧の比較動作を停止します。

[指定形式]

void	R_COMP*n*_Stop (void);

備考 *n* は、チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_PGA_Start

プログラマブル・ゲイン・アンプの動作を開始します。

[指定形式]

```
void        R_PGA_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_PGA_Stop

プログラマブル・ゲイン・アンプの動作を停止します。

[指定形式]

```
void        R_PGA_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_PWMOPT_Start

6 相 PWM オプション・ユニットに対する入力クロックを供給します。
また、6 相 PWM オプション・ユニットの動作モードを設定します。

[指定形式]

```
void        R_PWMOPT_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_PWMOPT_Stop

6 相 PWM オプション・ユニットに対するクロック供給を停止します。

[指定形式]

```
void        R_PWMOPT_Stop ( void );
```

[引数]

なし

[戻り値]

なし

3.3.29 シリアル・アレイ・ユニット

以下に、コード生成ツールがシリアル・アレイ・ユニット用として出力する API 関数の一覧を示します。

表 3.30 シリアル・アレイ・ユニット用 API 関数（1）

API 関数名	機能概要
R_SAUm_Create	シリアル・アレイ・ユニッ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_SAUm_Create_UserInit	シリアル・アレイ・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_SAUm_Reset	シリアル・アレイ・ユニットをリセットします。
R_SAUm_Set_PowerOff	シリアル・アレイ・ユニットに対するクロック供給を停止します。
R_SAUm_Set_SnoozeOn	STOP モードから SNOOZE モードへの切り替えを許可します。
R_SAUm_Set_SnoozeOff	STOP モードから SNOOZE モードへの切り替えを禁止します。
R_UARTn_Create	UART 通信を行ううえで必要となる初期化処理を行います。
r_uartn_interrupt_send	UART 送信完了割り込み INTST*n* の発生に伴う処理を行います。
r_uartn_interrupt_receive	UART 受信完了割り込み INTSR*n* の発生に伴う処理を行います。
r_uartn_callback_error	受信エラー割り込み INTSRE*n* の発生に伴う処理を行います。
R_UARTn_Start	UART 通信を待機状態にします。
R_UARTn_Stop	UART 通信を終了します。
R_UARTn_Send	データの UART 送信を開始します。
R_UARTn_Receive	データの UART 受信を開始します。
r_uartn_callback_sendend	UART 送信完了割り込み INTST*n* の発生に伴う処理を行います。
r_uartn_callback_receiveend	UART 受信完了割り込み INTSR*n* の発生に伴う処理を行います。
r_uartn_callback_error	UART 受信エラー割り込み INTSRE*n* の発生に伴う処理を行います。
r_uartn_callback_softwareoverrun	オーバラン・エラーの検出に伴う処理を行います。
R_CSImn_Create	3 線シリアル I/O 通信を行ううえで必要となる初期化処理を行います。
r_csimn_interrupt	CSI 通信完了割り込み INTCSI*mn* の発生に伴う処理を行います。
R_CSImn_Start	3 線シリアル I/O 通信を待機状態にします。
R_CSImn_Stop	3 線シリアル I/O 通信を終了します。
R_CSImn_Send	データの CSI 送信を開始します。
R_CSImn_Receive	データの CSI 受信を開始します。
R_CSImn_Send_Receive	データの CSI 送受信を開始します。
r_csimn_callback_sendend	CSI 送信完了割り込み INTCSI*mn* の発生に伴う処理を行います。
r_csimn_callback_receiveend	CSI 受信完了割り込み INTCSI*mn* の発生に伴う処理を行います。
r_csimn_callback_error	CSI 受信エラー割り込み INTSRE*n* の発生に伴う処理を行います。
R_IICmn_Create	簡易 IIC 通信を行ううえで必要となる初期化処理を行います。
r_iicmn_interrupt	簡易 IIC 通信完了割り込み INTIIC*mn* の発生に伴う処理を行います。
R_IICmn_StartCondition	スタート・コンディションを発生させます。
R_IICmn_StopCondition	ストップ・コンディションを発生させます。
R_IICmn_Stop	簡易 IIC 通信を終了します。

表 3.31 シリアル・アレイ・ユニット用 API 関数（2）

API 関数名	機能概要
R_IICmn_Master_Send	簡易 IIC マスタ送信を開始します。
R_IICmn_Master_Receive	簡易 IIC マスタ受信を開始します。
r_iicmn_callback_master_sendend	簡易 IIC マスタ送信完了割り込み INTIIC*mn* の発生に伴う処理を行います。
r_iicmn_callback_master_receiveend	簡易 IIC マスタ受信完了割り込み INTIIC*mn* の発生に伴う処理を行います。
r_iicmn_callback_master_error	パリティ・エラー（ACK エラー）の検出に伴う処理を行います。

R_SAU*m*_Create

シリアル・アレイ・ユニットを制御するうえで必要となる初期化処理を行います。

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

[指定形式]

void	R_SAU*m*_Create (void);

備考　*m* は、ユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_SAU*m*_Create_UserInit

シリアル・アレイ・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_SAUm_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_SAU*m*_Create_UserInit (void);

備考　*m* は、ユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_SAUm_Reset

シリアル・アレイ・ユニットをリセットします。

[指定形式]

void	R_SAU*m*_Reset (void);

備考 *m* は、ユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_SAU*m*_Set_PowerOff

シリアル・アレイ・ユニットに対するクロック供給を停止します。

備考　本 API 関数の呼び出しにより、シリアル・アレイ・ユニットはリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタ（シリアル・クロック選択レジスタ *n*：SPS*n* など）への書き込みは無視されます。

[指定形式]

void	R_SAU*m*_Set_PowerOff (void);

備考　*m* は、ユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_SAU*m*_Set_SnoozeOn

STOP モードから SNOOZE モードへの切り替えを許可します。

[指定形式]

void	R_SAU*m*_Set_SnoozeOn (void);

備考 *m* は、ユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_SAU*m*_Set_SnoozeOff

STOP モードから SNOOZE モードへの切り替えを禁止します。

[指定形式]

void	R_SAU*m*_Set_SnoozeOff (void);

備考 *m* は、ユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_UART*n*_Create

UART 通信を行う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、R_SAUm_Create の内部関数として位置づけられているため、通常、ユーザの処理プログラムから呼び出す必要はありません

[指定形式]

void	R_UART*n*_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_uart*n*_interrupt_send

UART 送信完了割り込み INTST*n* の発生に伴う処理を行います。

備考　本 API 関数は、UART 送信完了割り込み INTST*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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_uartn_interrupt_send ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_uartn_interrupt_send ( 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_uart*n*_interrupt_receive

UART 受信完了割り込み INTSR*n* の発生に伴う処理を行います。

備考　本 API 関数は、UART 受信完了割り込み INTSR*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]

CA78K0R コンパイラの場合

```
__interrupt static void    r_uartn_interrupt_receive ( void );
```

CC-RL コンパイラの場合

```
static void    __near r_uartn_interrupt_receive ( 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_uart*n*_interrupt_error

受信エラー割り込み INTSRE*n* 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 INTSRE*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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_uartn_interrupt_error ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_uartn_interrupt_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

R_UART*n*_Start

UART 通信を待機状態にします。

[指定形式]

void	R_UART*n*_Start (void);

備考 *n* は、チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_UART*n*_Stop

UART 通信を終了します。

[指定形式]

void	R_UART*n*_Stop (void);

備考 *n* は、チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_UART*n*_Send

データの UART 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 UART 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. UART 送信を行う際には、本 API 関数の呼び出し以前に R_UARTn_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_UARTn_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_UART*n*_Receive

データの UART 受信を開始します。

備考 1. 本 API 関数では、 1 バイト単位の UART 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. UART 受信を行う際には、本 API 関数の呼び出し以前に R_UARTn_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_UART*n*_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	引数の指定が不正

r_uart*n*_callback_sendend

UART 送信完了割り込み INTST*n* の発生に伴う処理を行います。

備考　本 API 関数は、UART 送信完了割り込み INTST*n* に対応した割り込み処理 r_uartn_interrupt_send のコールバック・ルーチン（R_UARTn_Send の引数 *tx_num* で指定された数のデータ送信が完了した際の処理）として呼び出されます。

[指定形式]

```
static void    r_uartn_callback_sendend ( 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_uart*n*_callback_receiveend

UART 受信完了割り込み INTSR*n* の発生に伴う処理を行います。

備考 本 API 関数は、UART 受信完了割り込み INTSR*n* に対応した割り込み処理 r_uartn_interrupt_receive のコールバック・ルーチン（R_UARTn_Receive の引数 *rx_num* で指定された数のデータ受信が完了した際の処理）として呼び出されます。

[指定形式]

```
static void    r_uartn_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_uart*n*_callback_error

UART 受信エラー割り込み INTSRE*n* の発生に伴う処理を行います。

備考 本 API 関数は、UART 通信エラー割り込み INTSRE*n* に対応した割り込み処理 r_uartn_interrupt_error 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_uart*n*_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*;	UART 受信エラー割り込みの発生要因 00000xx1B : オーバラン・エラー 00000x1xB : パリティ・エラー 000001xxB : フレーミング・エラー

[戻り値]

なし

r_uart*n*_callback_softwareoverrun

オーバラン・エラーの検出に伴う処理を行います。

備考 本 API 関数は、UART 受信完了割り込み INTSR*n* に対応した割り込み処理 r_uartn_interrupt_receive のコールバック・ルーチン（R_UARTn_Receive の引数 *rx_num* で指定された数以上のデータを受信した際の処理）として呼び出されます。

[指定形式]

```
#include      “r_cg_macrodriver.h”
static void    r_uartn_callback_softwareoverrun ( uint16_t rx_data );
```

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	uint16_t *rx_data*;	受信したデータ（R_UARTn_Receive の引数 *rx_num* で指定された数以上に受信したデータ）

[戻り値]

なし

R_CSI*mn*_Create

3 線シリアル I/O 通信を行う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、R_SAUm_Create の内部関数として位置づけられているため、通常、ユーザの処理プログラムから呼び出す必要はありません

[指定形式]

void	R_CSI*mn*_Create (void);

備考 *m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

r_csi*mn*_interrupt

CSI 通信完了割り込み INTCSI*mn* の発生に伴う処理を行います。

備考　本 API 関数は、CSI 通信完了 INTCSI*mn* 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]

CA78K0R コンパイラの場合

```
__interrupt static void  r_csimn_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void  __near r_csimn_interrupt ( void );
```

備考　*m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_CSI*mn*_Start

3 線シリアル I/O 通信を待機状態にします。

[指定形式]

void	R_CSI*mn*_Start (void);

備考 *m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_CSI*mn*_Stop

3 線シリアル I/O 通信を終了します。

[指定形式]

void	R_CSI*mn*_Stop (void);

備考 *m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_CSI*mn*_Send

データの CSI 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 CSI 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. CSI 送信を行う際には、本 API 関数の呼び出し以前に R_CSImn_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_CSImn_Send ( uint8_t * const tx_buf, uint16_t tx_num );
```

備考 *m* はユニット番号を、*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_CSI*mn*_Receive

データの CSI 受信を開始します。

備考 1. 本 API 関数では、 1 バイト単位の CSI 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. CSI 受信を行う際には、本 API 関数の呼び出し以前に R_CSImn_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_CSImn_Receive ( uint8_t * const rx_buf, uint16_t rx_num );
```

備考 *m* はユニット番号を、*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	引数の指定が不正

R_CSI*mn*_Send_Receive

データの CSI 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 CSI 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 *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. CSI 送受信を行う際には、本 API 関数の呼び出し以前に R_CSImn_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_CSImn_Send_Receive ( uint8_t * const tx_buf, uint16_t tx_num,
 uint8_t * const rx_buf );
```

備考 *m* はユニット番号を、*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_csi*mn*_callback_sendend

CSI 送信完了割り込み INTCSI*mn* の発生に伴う処理を行います。

備考 1. 本 API 関数は、CSI 送信完了割り込み INTCSI*mn* に対応した割り込み処理 r_csimn_interrupt のコールバック・ルーチン（R_CSImn_Send、または R_CSImn_Send_Receive の引数 *tx_num* で指定された数のデータ送信が完了した際の処理）として呼び出されます。

備考 2. 連続モードで、送信または送受信を繰り返し実施する場合には、連続モードに再設定するためにコールバック関数に下記を追記していください。

- r_csimn_callback_sendend(): “|= _0001_SAU_BUFFER_EMPTY;”

[指定形式]

```
static void    r_csimn_callback_sendend ( void );
```

備考 *m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

r_csi*mn*_callback_receiveend

CSI 受信完了割り込み INTCSI*mn* の発生に伴う処理を行います。

備考 1. 本 API 関数は、CSI 受信完了割り込み INTCSI*mn* に対応した割り込み処理 r_csimn_interrupt のコールバック・ルーチン（R_CSImn_Receive の引数 *rx_num*、または R_CSImn_Send_Receive の引数 *tx_num* で指定された数のデータ受信が完了した際の処理）として呼び出されます。

備考 2. 連続モードで、送信または送受信を繰り返し実施する場合には、連続モードに再設定するためにコールバック関数に下記を追記していください。

- r_csimn_callback_receiveend(): “SMRmn |= _0001_SAU_BUFFER_EMPTY;”

[指定形式]

```
static void    r_csin_callback_receiveend ( void );
```

備考 *m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

r_csi*mn*_callback_error

CSI 受信エラー割り込み INTSRE*n* の発生に伴う処理を行います。

備考　本 API 関数は、CSI 受信エラー割り込み INTSRE*n* に対応した割り込み処理 r_uartn_interrupt_error 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_csimn_callback_error ( uint8_t err_type );
```

備考　*m* はユニット番号を、*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*;	CSI 受信エラー割り込みの発生要因 00000xx1B：　オーバラン・エラー

[戻り値]

なし

R_IIC*mn*_Create

簡易 IIC 通信を行う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、R_SAUm_Create の内部関数として位置づけられているため、通常、ユーザの処理プログラムから呼び出す必要はありません

[指定形式]

void	R_IICI*n*_Create (void);

備考 *m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

r_iic*mn*_interrupt

簡易 IIC 転送完了割り込み INTIIC*mn* の発生に伴う処理を行います。

備考 1. 本 API 関数は、簡易 IIC 転送完了割り込み INTIICmn 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、ストップ・コンディションは生成されません。

[指定形式]

CA78K0R コンパイラの場合

```
__interrupt static void     r_iicmn_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_iicmn_interrupt ( void );
```

備考 *m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_IIC*mn*_StartCondition

スタート・コンディションを発生させます。

備考 本 API 関数は、R_IIC*mn*_Master_Send、および R_IIC*mn*_Master_Receive の内部関数として位置づけられているため、通常、ユーザの処理プログラムから呼び出す必要はありません。

[指定形式]

void	R_IIC*mn*_StartCondition (void);

備考 *m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_IIC*mn*_StopCondition

ストップ・コンディションを発生します。

備考 1. ユーザはスレーブ処理の完了を確認し、main() 関数でストップ・コンディションを設定する必要があります。

備考 2. 割り込み処理 r_iic*mn*_interrupt, コールバック・ルーチン r_iic*mn*_callback_master_sendend, r_iic*mn*_callback_master_receiveend, r_iic*mn*_callback_master_error で設定しないでください。

[指定形式]

void	R_SCI*n*_IIC_StopCondition (void);

備考 *m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_IIC*mn*_Stop

簡易 IIC 通信を終了します。

[指定形式]

void	R_IIC*mn*_Stop (void);

備考 *m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_IIC*mn*_Master_Send

簡易 IIC マスタ送信を開始します。

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 バイト単位の簡易 IIC 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 関数を呼び出し以前に、通信動作停止状態または通信動作待機状態、および SDA/SCL 端子レベルがハイ・レベルであることを確認してください。

[指定形式]

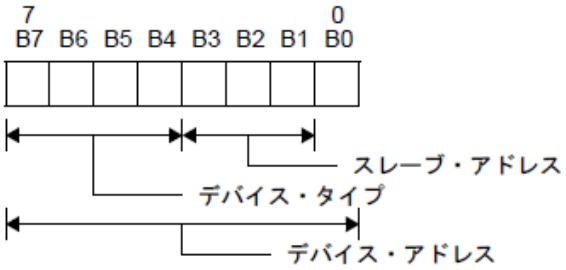
```
#include   “r_cg_macrodriver.h”
void       R_IIC mn _Master_Send ( uint8_t adr, uint8_t * const tx_buf, uint16_t tx_num );
```

備考 *m* はユニット番号を、*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*;	デバイス・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

備考 以下に、デバイス・アドレス adr の指定形式を示します。



[戻り値]

なし

R_IIC*mn*_Master_Receive

簡易 IIC マスタ受信を開始します。

備考 1. 本 API 関数では、1 バイト単位の簡易 IIC 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 関数を呼び出し以前に、通信動作停止状態または通信動作待機状態、および SDA/SCL 端子レベルがハイ・レベルであることを確認してください。

[指定形式]

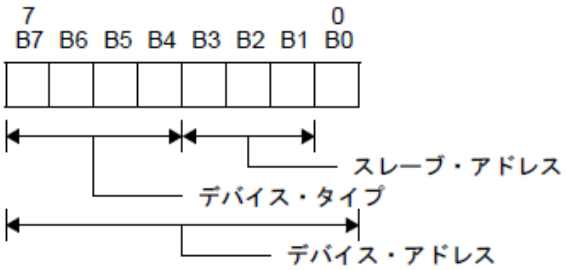
```
#include  “r_cg_macrodriver.h”
void      R_IIC mn _Master_Receive ( uint8_t adr, uint8_t * const rx_buf, uint16_t rx_num );
```

備考 *m* はユニット番号を、*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

備考 以下に、デバイス・アドレス adr の指定形式を示します。



[戻り値]

なし

r_iic*mn*_callback_master_sendend

簡易 IIC 転送完了（マスタ送信完了）割り込み INTIIC*mn* の発生に伴う処理を行います。

備考　本 API 関数は、簡易 IIC 転送完了（マスタ送信完了）割り込み INTIIC*mn* に対応した割り込み処理 r_iic*mn*_interrupt のコールバック・ルーチン（R_IIC*mn*_Master_Send の引数 *tx_num* で指定された数のデータ送信が完了した際の処理）として呼び出されます。

[指定形式]

```
static void    r_iicmn_callback_master_sendend ( void );
```

備考　*m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

r_iic*mn*_callback_master_receiveend

簡易 IIC 転送完了（マスタ受信完了）割り込み INTIIC*mn* の発生に伴う処理を行います。

備考　本 API 関数は、簡易 IIC 転送完了（マスタ受信完了）割り込み INTIIC*mn* に対応した割り込み処理 r_iic*mn*_interrupt のコールバック・ルーチン（R_IIC*mn*_Master_Receive の引数 *rx_num* で指定された数のデータ受信が完了した際の処理）として呼び出されます。

[指定形式]

```
static void    r_iicn_callback_master_receiveend ( void );
```

備考　*m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

r_iic*mn*_callback_master_error

ACK エラーまたはオーバラン・エラーの検出に伴う処理を行います。

備考 本 API 関数は、簡易 IIC 転送完了割り込み INTIIC*mn* に対応した割り込み処理 r_iic*mn*_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_iicmn_callback_master_error ( MD_STATUS flag );
```

備考 *m* はユニット番号を、*n* はチャネル番号を意味します。

[引数]

I/O	引数	説明
O	MD_STATUS *flag*;	通信エラーの発生要因 MD_NACK： アクノリッジの未検出 MD_OVERRUN： オーバラン・エラーの検出

[戻り値]

なし

使用例

UART 通信で 4Byte 受信後、受信したデータをそのまま送信し、通信を終了させる

[GUI 設定例]

シリアル				使用する
	SAU0			使用する
		Channel 0		
			チャネル 0	UART0(送信/受信機能)
			データ・ビット長設定 (受信機能)	8 ビット
			データ転送方向設定 (受信機能)	LSB
			パリティ設定 (受信機能)	パリティなし
			ストップ・ビット長設定 (受信機能)	1 ビット固定です
			受信データ・レベル設定	標準
			転送レート設定 (受信機能)	9600(bps)(誤差 : +0.16% 許容最小 : -5.17% 許容最大 : +5.16%)
			受信完了割り込み設定(INTSR0)	低
			受信完了 (コールバック機能設定)	使用する
			エラー (コールバック機能設定)	使用する
			転送モード設定	単発モード
			データ・ビット長設定 (送信機能)	8 ビット
			データ転送方向設定 (送信機能)	LSB
			パリティ設定 (送信機能)	パリティなし
			ストップ・ビット長設定 (送信機能)	1 ビット
			送信データ・レベル設定	標準
			転送レート設定 (送信機能)	9600(bps)(誤差 : +0.16%)
			送信完了割り込み設定(INTST0)	低
			送信完了 (コールバック機能設定)	使用する

[API 設定例]

r_main.c

```
/* Start user code for global. Do not edit comment generated here */
extern volatile uint8_t g_uart0_buf[4];
/* End user code. Do not edit comment generated here */

void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start the UART0 module operation */
    R_UART0_Start();

    /* Receive UART0 data */
    R_UART0_Receive((uint8_t *)g_uart0_buf, 4U);

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

r_cg_serial_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_uart0_buf[4];
/* End user code. Do not edit comment generated here */

static void r_uart0_callback_receiveend(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Send UART0 data */
    R_UART0_Send((uint8_t *)g_uart0_buf, 4U);
    /* End user code. Do not edit comment generated here */
}

static void r_uart0_callback_sendend(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop the UART0 module operation */
    R_UART0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

3.3.30 シリアル・アレイ・ユニット 4

以下に、コード生成ツールがシリアル・アレイ・ユニット 4 用として出力する API 関数の一覧を示します。

表 3.32 シリアル・アレイ・ユニット 4 用 API 関数

API 関数名	機能概要
R_DALIn_Create	シリアル・アレイ・ユニット 4 を制御するうえで必要となる初期化処理を行います。
r_dalin_interrupt_send	DALI 送信完了割り込み INTSTDL*n* の発生に伴う処理を行います。
r_dalin_interrupt_receive	DALI 受信完了割り込み INTSRDL*n* の発生に伴う処理を行います。
r_dalin_interrupt_error	DALI 受信エラー割り込み INTSREDL*n* の発生に伴う処理を行います。
R_DALIn_Start	DALI 通信を待機状態にします。
R_DALIn_Stop	DALI 通信を終了します。
R_DALIn_Send	データの DALI 送信を開始します。
R_DALIn_Receive	データの DALI 受信を開始します。
r_dalin_callback_sendend	DALI 送信完了割り込み INTSTDL*n* の発生に伴う処理を行います。
r_dalin_callback_receiveend	DALI 受信完了割り込み INTSRDL*n* の発生に伴う処理を行います。
r_dalin_callback_error	DALI 受信エラー割り込み INTSREDL*n* の発生に伴う処理を行います。
r_dalin_callback_softwareoverrun	オーバラン・エラーの検出に伴う処理を行います。

R_DALI*n*_Create

シリアル・アレイ・ユニット 4 を制御するうえで必要となる初期化処理を行います。

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

[指定形式]

void	R_DALI*n*_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_dali*n*_interrupt_send

DALI 送信完了割り込み INTSTDL*n* の発生に伴う処理を行います。

備考　本 API 関数は、DALI 送信完了割り込み INTSTDL*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]

CA78K0R コンパイラの場合

```
__interrupt static void    r_dali n_interrupt_send ( void );
```

CC-RL コンパイラの場合

```
static void    __near r_dali n_interrupt_send ( 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_dali*n*_interrupt_receive

DALI 受信完了割り込み INTSRDL*n* の発生に伴う処理を行います。

備考 本 API 関数は、DALI 受信完了割り込み INTSRDL*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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_dali n_interrupt_receive ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_dali n_interrupt_receive ( 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_dali*n*_interrupt_error

DALI 受信エラー割り込み INTSREDL*n* の発生に伴う処理を行います。

備考　本 API 関数は、DALI 受信エラー割り込み INTSREDL*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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_dali n_interrupt_error ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_dali n_interrupt_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

R_DALI*n*_Start

DALI 通信を待機状態にします。

[指定形式]

void	R_DALI*n*_Start (void);

備考 *n* は、チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_DALI*n*_Stop

DALI 通信を終了します。

[指定形式]

void	R_DALI*n*_Stop (void);

備考 *n* は、チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_DALI*n*_Send

データの DALI 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 DALI 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. DALI 送信を行う際には、本 API 関数の呼び出し以前に R_DALIn_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_DALIn_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_DALI*n*_Receive

データの DALI 受信を開始します。

備考 1. 本 API 関数では、 1 バイト単位の DALI 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. DALI 受信を行う際には、本 API 関数の呼び出し以前に R_DALIn_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_DALIn_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	引数の指定が不正

r_dali*n*_callback_sendend

DALI 送信完了割り込み INTSTSL*n* の発生に伴う処理を行います。

備考 本 API 関数は、DALI 送信完了割り込み INTSTDL*n* に対応した割り込み処理 r_dalin_interrupt_send のコールバック・ルーチン（R_DALIn_Send の引数 *tx_num* で指定された数のデータ送信が完了した際の処理）として呼び出されます。

[指定形式]

```
static void    r_dalin_callback_sendend ( 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_dali*n*_callback_receiveend

DALI 受信完了割り込み INTSRDL*n* の発生に伴う処理を行います。

備考 本 API 関数は、DALI 受信完了割り込み INTSRDL*n* に対応した割り込み処理 r_dalin_interrupt_receive のコールバック・ルーチン（R_DALIn_Receive の引数 *rx_num* で指定された数のデータ受信が完了した際の処理）として呼び出されます。

[指定形式]

```
static void    r_dali*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_dali*n*_callback_error

DALI 受信エラー割り込み INTSREDL*n* の発生に伴う処理を行います。

備考　本 API 関数は、DALI 通信エラー割り込み INTSREDL*n* に対応した割り込み処理 r_dalin_interrupt_error 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_dalin_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*;	DALI 受信エラー割り込みの発生要因 00000xx1B :　オーバラン・エラー 00000x1xB :　パリティ・エラー 000001xxB :　フレーミング・エラー

[戻り値]

なし

r_dali*n*_callback_softwareoverrun

オーバラン・エラーの検出に伴う処理を行います。

備考　本 API 関数は、DALI 受信完了割り込み INTSRDL*n* に対応した割り込み処理 r_dalin_interrupt_receive のコールバック・ルーチン（R_DALIn_Receive の引数 *rx_num* で指定された数以上のデータを受信した際の処理）として呼び出されます。

[指定形式]

```
#include      “r_cg_macrodriver.h”
static void    r_dali n_callback_softwareoverrun ( uint16_t rx_data );
```

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	uint16_t *rx_data*;	受信したデータ（R_DALIn_Receive の引数 *rx_num* で指定された数以上に受信したデータ）

[戻り値]

なし

使用例

UART 通信として使用する際は、「3.2.30 シリアル・アレイ・ユニットの使用例」を参照してください。

3.3.31 アシンクロナス・シリアル・インタフェース LIN-UART

以下に、コード生成ツールがアシンクロナス・シリアル・インタフェース LIN-UART 用として出力する API 関数の一覧を示します。

表 3.33 アシンクロナス・シリアル・インタフェース LIN-UART 用 API 関数

API 関数名	機能概要
R_UARTFn_Create	アシンクロナス・シリアル・インタフェース LIN-UART を制御するうえで必要となる初期化処理を行います。
R_UARTFn_Create_UserInit	アシンクロナス・シリアル・インタフェース LIN-UART に関するユーザ独自の初期化処理を行います。
r_uartfn_interrupt_send	LIN-UART 送信完了割り込み INTLT の発生に伴う処理を行います。
r_uartfn_interrupt_receive	LIN-UART 受信完了割り込み INTLR の発生に伴う処理を行います。
r_uartfn_interrupt_error	LIN-UART 受信ステータス割り込み INTLS の発生に伴う処理を行います。
R_UARTFn_Start	LIN 通信を待機状態にします。
R_UARTFn_Stop	LIN 通信を終了します。
R_UARTFn_Set_PowerOff	アシンクロナス・シリアル・インタフェース LIN-UART に対するクロック供給を停止します。
R_UARTFn_Send	データの UARTF 送信を開始します。
R_UARTFn_Receive	データの UARTF 受信を開始します。
R_UARTFn_Set_DataComparisonOn	データの比較を開始します。
R_UARTFn_Set_DataComparisonOff	データの比較を終了します。
r_uartfn_callback_sendend	LIN-UART 送信完了割り込み INTLT の発生に伴う処理を行います。
r_uartfn_callback_receiveend	LIN-UART 受信完了割り込み INTLR の発生に伴う処理を行います。
r_uartfn_callback_error	LIN-UART 受信ステータス割り込み INTLS の発生に伴う処理を行います。
r_uartfn_callback_softwareoverrun	オーバラン・エラーの検出に伴う処理を行います。
r_uartfn_callback_expbitdetect	拡張ビットの検出に伴う処理を行います。
r_uartfn_callback_idmatch	ID パリティの一致に伴う処理を行います。

R_UARTF*n*_Create

アシンクロナス・シリアル・インタフェース LIN-UART を制御するうえで必要となる初期化処理を行います。

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

[指定形式]

void	R_UARTF*n*_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_UARTF*n*_Create_UserInit

アシンクロナス・シリアル・インタフェース LIN-UART 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_UARTFn_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_UARTF*n*_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_uartf*n*_interrupt_send

LIN-UART 送信完了割り込み INTLT の発生に伴う処理を行います。

備考　本 API 関数は、LIN-UART 送信完了割り込み INTSTDL*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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_uartfn_interrupt_send ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_uartfn_interrupt_send ( 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_uartf*n*_interrupt_receive

LIN-UART 受信完了割り込み INTLR の発生に伴う処理を行います。

備考　本 API 関数は、LIN-UART 受信完了割り込み INTLR 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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_uartfn_interrupt_receive ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_uartfn_interrupt_receive ( 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_uartf*n*_interrupt_error

LIN-UART 受信ステータス割り込み INTLS の発生に伴う処理を行います。

備考　本 API 関数は、LIN-UART 受信ステータス割り込み INTLS 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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_uartfn_interrupt_error ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_uartfn_interrupt_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

R_UARTF*n*_Start

LIN 通信を待機状態にします。

[指定形式]

void	R_UARTF*n*_Start (void);

備考 *n* は、チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_UARTF*n*_Stop

LIN 通信を終了します。

[指定形式]

void	R_UARTF*n*_Stop (void);

備考 *n* は、チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_UARTF*n*_Set_PowerOff

アシンクロナス・シリアル・インタフェース LIN-UART に対するクロック供給を停止します。

備考 本 API 関数の呼び出しにより、アシンクロナス・シリアル・インタフェース LIN-UART はリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

void	R_UARTF*n*_Set_PowerOff (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_UARTF*n*_Send

データの UARTF 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 UARTF 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. UARTF 送信を行う際には、本 API 関数の呼び出し以前に R_UARTFn_Start を呼び出す必要があります。

備考 3. アシンクロナス・シリアル・インタフェース LIN-UART を拡張ビット・モードで使用する場合、送信するデータを以下の形式で、引数 *tx_buf* で指定した領域に格納してください。
“8 ビット・データ”, “拡張ビット”, “8 ビット・データ”, “拡張ビット”, …

[指定形式]

```
#include        “r_cg_macrodriver.h”
MD_STATUS    R_UARTFn_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
MD_DATAEXISTS	送信処理を実行中

R_UARTF*n*_Receive

データの UARTF 受信を開始します。

備考 1. 本 API 関数では、 1 バイト単位の UARTF 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. 実際の UARTF 受信は、本 API 関数の呼び出し後、R_UARTFn_Start を呼び出すことにより開始されます。

備考 3. アシンクロナス・シリアル・インタフェース LIN-UART を拡張ビット・モードで使用する場合、受信したデータは以下の形式で、引数 *rx_buf* で指定した領域に格納されます。
“8 ビット・データ”, “拡張ビット”, “8 ビット・データ”, “拡張ビット”, …

[指定形式]

```
#include        “r_cg_macrodriver.h”
MD_STATUS    R_UARTF_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	引数の指定が不正

R_UARTF*n*_Set_DataComparisonOn

データの比較を開始します。

備考　本 API 関数の呼び出しにより、アシンクロナス・シリアル・インタフェース LIN-UART は拡張ビット・モード（データ比較あり）へと移行します。

[指定形式]

void	R_UARTF*n*_Set_DataComparisonOn (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_UARTF*n*_Set_DataComparisonOff

データの比較を終了します。

備考　本 API 関数の呼び出しにより、アシンクロナス・シリアル・インタフェース LIN-UART は拡張ビット・モード（データ比較なし）へと移行します。

[指定形式]

void	R_UARTF*n*_Set_DataComparisonOff (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_uartf*n*_callback_sendend

LIN-UART 送信完了割り込み INTLT の発生に伴う処理を行います。

備考　本 API 関数は、LIN-UART 送信完了割り込み INTLT に対応した割り込み処理 r_uartfn_interrupt_send のコールバック・ルーチン（R_UARTFn_Send の引数 *tx_num* で指定された数のデータ送信が完了した際の処理）として呼び出されます。

[指定形式]

```
static void    r_uartfn_callback_sendend ( 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_uartf*n*_callback_receiveend

LIN-UART 受信完了割り込み INTLR の発生に伴う処理を行います。

備考 本 API 関数は、LIN-UART 受信完了割り込み INTLR に対応した割り込み処理 r_uartfn_interrupt_receive のコールバック・ルーチン（R_UARTFn_Receive の引数 *rx_num* で指定された数のデータ受信が完了した際の処理）として呼び出されます。

[指定形式]

```
static void    r_uartfn_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_uartf*n*_callback_error

LIN-UART 受信ステータス割り込み INTLS の発生に伴う処理を行います。

備考　本 API 関数は、LIN-UART 通信エラー割り込み INTLS に対応した割り込み処理 r_uartfn_interrupt_error 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_uartfn_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*;	LIN-UART 受信ステータス割り込みの発生要因 00000xx1B :　オーバラン・エラー 00000x1xB :　パリティ・エラー 000001xxB :　フレーミング・エラー

[戻り値]

なし

r_uartf*n*_callback_softwareoverrun

オーバラン・エラーの検出に伴う処理を行います。

備考 本 API 関数は、LIN-UART 受信完了割り込み INTLR に対応した割り込み処理 r_uartfn_interrupt_receive のコールバック・ルーチン（R_UARTFn_Receive の引数 *rx_num* で指定された数以上のデータを受信した際の処理）として呼び出されます。

[指定形式]

```
static void    r_uartfn_callback_softwareoverrun ( 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_uartf*n*_callback_expbitdetect

拡張ビットの検出に伴う処理を行います。

備考 本 API 関数は、LIN-UART 受信ステータス割り込み INTLS に対応した割り込み処理 r_uartfn_interrupt_error のコールバック・ルーチン（拡張ビットを検出した際の処理）として呼び出されます。

[指定形式]

```
static void    r_uartfn_callback_expbitdetect ( 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_uartf*n*_callback_idmatch

ID パリティの一致に伴う処理を行います。

備考 本 API 関数は、LIN-UART 受信ステータス割り込み INTLS に対応した割り込み処理 r_uartfn_interrupt_error のコールバック・ルーチン（ID パリティが一致した際の処理）として呼び出されます。

[指定形式]

```
static void    r_uartfn_callback_idmatch ( void );
```

備考 *n* は、チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

使用例

UART 通信として使用する際は、「3.2.30 シリアル・アレイ・ユニットの使用例」を参照してください。

3.3.32 シリアル・インタフェース IICA

以下に，コード生成ツールがシリアル・インタフェース IICA 用として出力する API 関数の一覧を示します。

表 3.34 シリアル・インタフェース IICA 用 API 関数

API 関数名	機能概要
R_IICAn_Create	シリアル･インタフェース IICA を制御するうえで必要となる初期化処理を行います。
R_IICAn_Create_UserInit	シリアル･インタフェース IICA に関するユーザ独自の初期化処理を行います。
r_iican_interrupt	IICA 通信完了割り込み INTIICA*n* の発生に伴う処理を行います。
R_IICAn_StopCondition	ストップコンディションを発生させます。
R_IICAn_Stop	IICA 通信を終了します。
R_IICAn_Reset	シリアル・インタフェース IICA をリセットします。
R_IICAn_Set_PowerOff	シリアル･インタフェース IICA に対するクロック供給を停止します。
R_IICAn_Master_Send	IICA マスタ送信を開始します。
R_IICAn_Master_Receive	IICA マスタ受信を開始します。
r_iican_callback_master_sendend	IICA マスタ送信完了割り込み INTIICA*n* の発生に伴う処理を行います。
r_iican_callback_master_receiveend	IICA マスタ受信完了割り込み INTIICA*n* の発生に伴う処理を行います。
r_iican_callback_master_error	IICA マスタ通信エラーの検出に伴う処理を行います。
R_IICAn_Slave_Send	IICA スレーブ送信を開始します。
R_IICAn_Slave_Receive	IICA スレーブ受信を開始します。
r_iican_callback_slave_sendend	IICA スレーブ送信完了割り込み INTIICA*n* の発生に伴う処理を行います。
r_iican_callback_slave_receiveend	IICA スレーブ受信完了割り込み INTIICA*n* の発生に伴う処理を行います。
r_iican_callback_master_error	IICA スレーブ通信エラーの検出に伴う処理を行います。
r_iican_callback_getstopcondition	ストップ・コンディションの検出に伴う処理を行います。
R_IICAn_Set_SnoozeOn	STOP モード時のアドレス一致ウェイクアップ機能の動作を許可します。
R_IICAn_Set_SnoozeOff	STOP モード時のアドレス一致ウェイクアップ機能の動作を禁止します。
R_IICAn_Set_WakeupOn	STOP モード時のアドレス一致ウェイクアップ機能の動作を許可します。
R_IICAn_Set_WakeupOff	STOP モード時のアドレス一致ウェイクアップ機能の動作を禁止します。

R_IICA*n*_Create

シリアル・インタフェース IICA を制御するうえで必要となる初期化処理を行います。

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

[指定形式]

void	R_IICA*n*_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_IICA*n*_Create_UserInit

シリアル・インタフェース IICA 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_IICAn_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_IICA*n*_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_iica*n*_interrupt

IICA 通信完了割り込み INTIICA*n* の発生に伴う処理を行います。

備考　本 API 関数は、IICA 通信完了割り込み INTIICA*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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_iica n_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_iica 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_IICA*n*_StopCondition

ストップ・コンディションを発生させます。

備考　本 API 関数の呼び出し後 IICA の動作を停止する前に、SPD0 ビットでストップコンディション検出されたことを確認してください。

[指定形式]

void	R_IICA*n*_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_IICA*n*_Stop

IICA 通信を終了します。

[指定形式]

void	R_IICA*n*_Stop (void);

備考 *n* は、チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_IICA*n*_Reset

シリアル・インタフェース IICA をリセットします。

[指定形式]

void	R_IICA*n*_Reset (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_IICA*n*_Set_PowerOff

シリアル・インタフェース IICA に対するクロック供給を停止します。

備考 本 API 関数の呼び出しにより、シリアル・インタフェース IICA はリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

void	R_IICA*n*_Set_PowerOff (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_IICA*n*_Master_Send

IICA マスタ送信を開始します。

備考　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 IICA 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。

[指定形式]

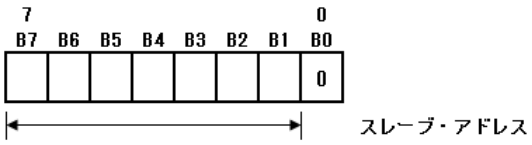
```
#include          “r_cg_macrodriver.h”
MD_STATUS     R_IICAn_Master_Send ( uint8_t adr, uint8_t * const tx_buf,
 uint16_t tx_num, uint8_t wait );
```

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
I	uint8_t *wait*	スタート・コンディションのセットアップ時間

備考　以下に、スレーブ・アドレス adr の指定形式を示します。
上位 7 ビットにスレーブ・アドレスを指定してください。本 API 関数内で最下位ビットを 0 に設定します。



[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR1	バス通信状態
MD_ERROR2	スタート・コンディション未検出

R_IICA*n*_Master_Receive

IICA マスタ受信を開始します。

備考 本 API 関数では、1 バイト単位の IICA 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。

[指定形式]

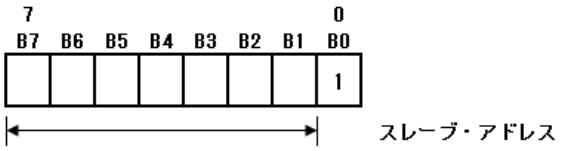
```
#include “r_cg_macrodriver.h”
MD_STATUD R_IICAn_Master_Receive ( uint8_t adr, uint8_t * const rx_buf,
 uint16_t rx_num, uint8_t wait );
```

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
I	uint8_t *wait*	スタート・コンディションのセットアップ時間

備考 以下に、スレーブ・アドレス adr の指定形式を示します。
上位 7 ビットにスレーブ・アドレスを指定してください。本 API 関数内で最下位ビットを 1 に設定します。



[戻り値]

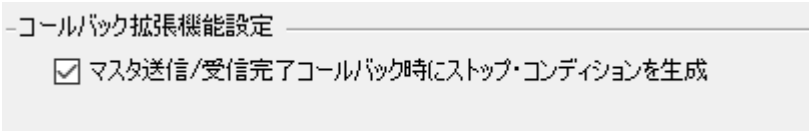
マクロ	説明
MD_OK	正常終了
MD_ERROR1	バス通信状態
MD_ERROR2	スタート・コンディション未検出

r_iica*n*_callback_master_sendend

IICA マスタ送信完了割り込み INTIICA*n* の発生に伴う処理を行います。

備考 1. 本 API 関数は、IICA マスタ送信完了割り込み INTIICA*n* に対応し割り込み処理 r_iican_interrupt のコールバック・ルーチンとして呼び出されます。

備考 2. 本 API 関数内に R_IICA*n*_StopCondition が生成／生成されないが、IICA の GUI 設定によって決まります。



チェック時：R_IICA*n*_StopCondition が本 API 関数内に生成されます。
未チェック時：R_IICA*n*_StopCondition が本 API 関数内に生成されません。

[指定形式]

```
static void    r_iican_callback_master_sendend ( void );
```

備考 *n* は、チャネル番号を意味します。

[引数]

なし

[戻り値]

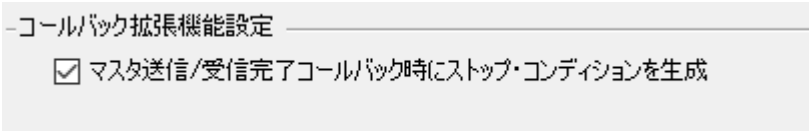
なし

r_iica*n*_callback_master_receiveend

IICA マスタ受信完了割り込み INTIICA*n* の発生に伴う処理を行います。

備考 1. 本 API 関数は、IICA マスタ受信完了割り込み INTIICA*n* に対応した割り込み処理 r_iican_interrupt のコールバック・ルーチンとして呼び出されます。

備考 2. 本 API 関数内に R_IICA*n*_StopCondition が生成／生成されないが、IICA の GUI 設定によって決まります。



チェック時：R_IICA*n*_StopCondition が本 API 関数内に生成されます。
未チェック時：R_IICA*n*_StopCondition が本 API 関数内に生成されません。

[指定形式]

```
static void    r_iican_callback_master_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_iica*n*_callback_master_error

IICA マスタ通信エラーの検出に伴う処理を行います。

[指定形式]

```
#include      “r_cg_macrodriver.h”
static void   r_iica n_callback_master_error ( MD_STATUS flag );
```

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 *flag*;	通信エラーの発生要因 MD_SPT：　ストップ・コンディションの検出 MD_NACK：　アクノリッジの未検出 （アドレス一致のスレーブがない、またはスレーブがデータ受信できない／次のデータを必要としない場合）

[戻り値]

なし

R_IICA*n*_Slave_Send

IICA スレーブ送信を開始します。

備考　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 IICA 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。

[指定形式]

```
#include  “r_cg_macrodriver.h”
void      R_IICAn_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_IICA*n*_Slave_Receive

IICA スレーブ受信を開始します。

備考 本 API 関数では、1 バイト単位の IICA 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。

[指定形式]

```
#include   “r_cg_macrodriver.h”
void       R_IICAn_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_iica*n*_callback_slave_sendend

IICA スレーブ送信完了割り込み INTIICA*n* の発生に伴う処理を行います。

備考　本 API 関数は、IICA スレーブ送信完了割り込み INTIICA*n* に対応し割り込み処理 r_iican_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_iica*n*_callback_master_sendend ( 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_iica*n*_callback_slave_receiveend

IICA スレーブ受信完了割り込み INTIICA*n* の発生に伴う処理を行います。

備考　本 API 関数は、IICA スレーブ受信完了割り込み INTIICA*n* に対応した割り込み処理 r_iican_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_iica*n*_callback_slave_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_iica*n*_callback_slave_error

IICA スレーブ通信エラーの検出に伴う処理を行います。

[指定形式]

```
#include      “r_cg_macrodriver.h”
static void    r_iica n_callback_slave_error ( MD_STATUS flag );
```

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 *flag*;	通信エラーの発生要因 MD_ERROR：　アドレス不一致の検出 MD_NACK：　アクノリッジの未検出 （マスタ受信終了）

[戻り値]

なし

r_iica*n*_callback_getstopcondition

ストップ・コンディションの検出に伴う処理を行います。

[指定形式]

```
static void     r_iica*n*_callback_getstopcondition ( 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_IICA*n*_Set_SnoozeOn

STOP モード時のアドレス一致ウェイクアップ機能の動作を許可します。

[指定形式]

void	R_IICA*n*_Set_SnoozeOn (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_IICA*n*_Set_SnoozeOff

STOP モード時のアドレス一致ウェイクアップ機能の動作を禁止します。

[指定形式]

void	R_IICA*n*_Set_SnoozeOff (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_IICA*n*_Set_WakeupOn

STOP モード時のアドレス一致ウェイクアップ機能の動作を許可します。

[指定形式]

void	R_IICA*n*_Set_WakeupOn (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_IICA*n*_Set_WakeupOff

STOP モード時のアドレス一致ウェイクアップ機能の動作を禁止します。

[指定形式]

void	R_IICA*n*_Set_WakeupOff (void);

備考 *n* は、チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

使用例(マスタ送信)

4Byte データのマスタ送信を行う

[GUI 設定例]

シリアル				使用する
	IICA0			使用する
			転送モード	シングルマスタ
		Master0		
			カウント・クロック設定	fCLK/2
			自局アドレス設定	16
			動作モード設定	標準
			転送クロック(fSCL)	100000(bps)(実際の値：99378.882)
			通信完了割り込み優先順位(INTIICA0)	高
			マスタ送信完了(コールバック機能設定)	使用する
			マスタ受信完了(コールバック機能設定)	使用しない
			マスタ・エラー(コールバック機能設定)	使用しない
			マスタ送信/受信完了コールバック時にストップ・コンディションを生成 (コールバック拡張機能設定)	使用する

[API 設定例]

r_main.c

```
/* Start user code for pragma. Do not edit comment generated here */
#define SLAVE_ADDR (0xA0)       /* slave address */
/* End user code. Do not edit comment generated here */

/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_iica0_tx_buf[4] = { 'A', 'B', 'C', 'D' };
/* End user code. Do not edit comment generated here */

void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start to send data as master mode */
    R_IICA0_Master_Send(SLAVE_ADDR, (uint8_t *)g_iica0_tx_buf, 4U, 128U);

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

r_cg_serial_user.c

```
static void r_iica0_callback_master_sendend(void)
{
    SPT0 = 1U;
    /* Start user code. Do not edit comment generated here */
    /* Stop IICA0 module operation */
    R_IICA0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

使用例(スレーブ受信)

4Byte データのスレーブ受信を行う

[GUI 設定例]

シリアル				使用する
	SAU0			使用しない
	SAU1			使用しない
	IICA0			使用する
			転送モード	スレーブ
		Slave0		
			カウント・クロック設定	fCLK/2
			自局アドレス設定	0xA0
			動作モード設定	標準
			ウェイクアップ機能設定	オフ
			通信完了割り込み優先順位(INTIICA0)	低
			スレーブ送信完了(コールバック機能設定)	使用しない
			スレーブ受信完了(コールバック機能設定)	使用する
			スレーブ・エラー(コールバック機能設定)	使用しない

[API 設定例]

r_main.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_iica0_rx_buf[4];
/* End user code. Do not edit comment generated here */

void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Receive data as slave mode */
    R_IICA0_Slave_Receive((uint8_t *)g_iica0_rx_buf, 4U);

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

r_cg_serial_user.c

```
static void r_iica0_callback_slave_receiveend(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop IICA0 module operation */
    R_IICA0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

使用例(マスタ受信)

4Byte データのマスタ受信を行う

[GUI 設定例]

シリアル				使用する
	IICA0			使用する
			転送モード	シングルマスタ
		Master0		
			カウント・クロック設定	fCLK/2
			自局アドレス設定	16
			動作モード設定	標準
			転送クロック(fSCL)	100000(bps)(実際の値：99378.882)
			通信完了割り込み優先順位(INTIICA0)	低
			マスタ送信完了(コールバック機能設定)	使用しない
			マスタ受信完了(コールバック機能設定)	使用する
			マスタ・エラー(コールバック機能設定)	使用しない
			マスタ送信/受信完了コールバック時にストップ・コンディションを生成 (コールバック拡張機能設定)	使用する

[API 設定例]

r_main.c

```
/* Start user code for pragma. Do not edit comment generated here */
#define SLAVE_ADDR (0xA0)       /* slave address */
/* End user code. Do not edit comment generated here */

/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_iica0_rx_buf[4];
/* End user code. Do not edit comment generated here */

void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start to receive IICA0 data as master mode */
    R_IICA0_Master_Receive(SLAVE_ADDR, (uint8_t *)g_iica0_rx_buf, 4U, 128U);

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

r_cg_serial_user.c

```
static void r_iica0_callback_master_receiveend(void)
{
    SPT0 = 1U;
    /* Start user code. Do not edit comment generated here */
    /* Stop IICA0 module operation */
    R_IICA0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

使用例(スレーブ送信)

4Byte データのスレーブ送信を行う

[GUI 設定例]

シリアル				使用する
	IICA0			使用する
			転送モード	スレーブ
		Slave0		
			カウント・クロック設定	fCLK/2
			自局アドレス設定	0xA0
			動作モード設定	標準
			ウェイクアップ機能設定	オフ
			通信完了割り込み優先順位(INTIICA0)	低
			スレーブ送信完了(コールバック機能設定)	使用する
			スレーブ受信完了(コールバック機能設定)	使用しない
			スレーブ･エラー(コールバック機能設定)	使用しない

[API 設定例]

r_main.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_iica0_tx_buf[4] = { 'A', 'B', 'C', 'D' };
/* End user code. Do not edit comment generated here */

void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Send data as slave mode */
    R_IICA0_Slave_Send((uint8_t *)g_iica0_tx_buf, 4U);

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

r_cg_serial_user.c

```
static void r_iica0_callback_slave_sendend(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop IICA0 module operation */
    R_IICA0_Stop();
    /* End user code. Do not edit comment generated here */
}
```

Config_RIIC0_user.c

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

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

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

        /* Send RIIC0 data to slave device */
        R_Config_RIIC0_Master_Send(0x00A0, (uint8_t *)g_riic0_tx_buf, 2U);
    }
    else
    {
        /* Stop the RIIC0 Bus Interface */
        R_Config_RIIC0_Stop();
    }
    /* End user code. Do not edit comment generated here */
}
```

3.3.33 LCD コントローラ／ドライバ

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 3.35 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_interrupt	LCD フレーム割り込み INTLCD の発生に伴う処理を行います。
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_Set_VoltageOn	内部昇圧回路、および容量分割回路を動作許可状態にします。
R_LCD_Set_VoltageOff	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_Set_PowerOff	LCD コントローラ／ドライバに対するクロック供給を停止します。
R_LCD_VoltageOn	内部昇圧回路、および容量分割回路を動作許可状態にします。
R_LCD_VoltageOff	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。

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

[指定形式]

```
void        R_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 );
```

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_lcd_interrupt

LCD フレーム割り込み INTLCD に伴う処理を行います。

備考　本 API 関数は、LCD フレーム割り込み INTLCD 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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_lcd_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_lcd_interrupt ( 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_Set_VoltageOn

内部昇圧回路、および容量分割回路を動作許可状態にします。

[指定形式]

```
void        R_LCD_Set_VoltageOn ( void );
```

[引数]

なし

[戻り値]

なし

R_LCD_Set_VoltageOff

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_Set_VoltageOff ( void );
```

[引数]

なし

[戻り値]

なし

R_LCD_Set_PowerOff

LCD コントローラ／ドライバに対するクロック供給を停止します。

備考 1. 本 API 関数の呼び出しにより、LCD コントローラ／ドライバはリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

備考 2. 本 API 関数では、周辺イネーブル・レジスタ *n* の RTCEN ビットを操作することにより、LCD コントローラ／ドライバに対するクロック供給の停止を実現しています。
このため、本 API 関数の呼び出しを行った際には、RTCEN ビットを共用している他の周辺装置（リアルタイム・クロックなど）に対するクロック供給も停止することになります。

[指定形式]

```
void        R_LCD_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

R_LCD_VoltageOn

内部昇圧回路、および容量分割回路を動作許可状態にします。

[指定形式]

void R_LCD_Set_VoltageOn (void);

[引数]

なし

[戻り値]

なし

R_LCD_VoltageOff

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_Set_VoltageOff ( void );
```

[引数]

なし

[戻り値]

なし

3.3.34 サウンド・ジェネレータ

以下に、コード生成ツールがサウンド・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 3.36 サウンド・ジェネレータ用 API 関数

API 関数名	機能概要
R_SG_Create	サウンド・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_SG_Create_UserInit	サウンド・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_sg_interrupt	対数元帥率のスレッシュ・ホールド値検出による割り込み INTSG の発生に伴う処理を行います。
R_SG_Start	サウンド・ジェネレータを動作許可状態にします。
R_SG_Stop	サウンド・ジェネレータを動作停止状態にします。

R_SG_Create

サウンド・ジェネレータを制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_SG_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_SG_Create_UserInit

サウンド・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_SG_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_SG_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_sg_interrupt

対数減衰率のスレッシュ・ホールド値検出による割り込み INTSG の発生に伴う処理を行います。

備考　本 API 関数は、対数減衰率のスレッシュ・ホールド値検出による割り込み INTSG 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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_sg_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_sg_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_SG_Start

サウンド・ジェネレータを動作許可状態にします。

[指定形式]

```
void        R_SG_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_SG_Stop

サウンド・ジェネレータを動作禁止状態にします。

[指定形式]

```
void        R_SG_Stop ( void );
```

[引数]

なし

[戻り値]

なし

3.3.35 DMA コントローラ

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

表 3.37 DMA コントローラ用 API 関数

API 関数名	機能概要
R_DMACn_Create	DMA コントローラを制御するうえで必要となる初期化処理を行います。
R_DMACn_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。
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_dmacn_interrupt	DMA 転送終了割り込み INTDMA*n* の発生に伴う処理を行います。
R_DMACn_Start	チャネル *n* を動作許可状態に設定します。
R_DMACn_Stop	チャネル *n* を動作停止状態に設定します。
R_DMACn_Set_SoftwareTriggerOn	DMA 転送を開始します。

R_DMAC*n*_Create

DMA コントローラを制御するうえで必要となる初期化処理を行います。

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

[指定形式]

void	R_DMAC*n*_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_DMAC*n*_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_DMACn_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*n*_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_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。

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

[指定形式]

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

[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

r_dmac*n*_interrupt

DMA 転送終了割り込み INTDMA*n* の発生に伴う処理を行います。

備考　本 API 関数は、DMA 転送終了割り込み INTDMA*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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_dmacn_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_dmacn_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*_Start

チャネル *n* を動作許可状態に設定します。

[指定形式]

void	R_DMAC*n*_Start (void);

備考 *n* は、チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_DMAC*n*_Stop

チャネル *n* を動作停止状態に設定します。

備考 1. 本 API 関数は、DMA 転送を強制終了させるものではありません。

備考 2. 本 API 関数は、“転送終了”の確認後に呼び出す必要があります。

[指定形式]

void	R_DMAC*n*_Stop (void);

備考 *n* は、チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_DMAC*n*_Set_SoftwareTriggerOn

DMA 転送を開始します。

[指定形式]

void	R_DMAC*n*_Set_SoftwareTriggerOn (void);

備考 *n* は、チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

使用例

AD 変換完了でデータ転送を開始する
（4 端子の AD 変換結果を、アドレスを移動させて各 4 データを格納し、平均値を求める）

[GUI 設定例]

DMA コントローラ				使用する
	DMA0			使用する
			DMA 動作設定	使用する
			転送方向設定	SFR → 内蔵 RAM
			転送データ・サイズ設定	8 ビット
			SFR アドレス	ADCR - 0x000fff1e
			RAM アドレス	0xffe00
			転送回数	4
			トリガ信号	INTAD(INTAD を設定してください)
			DMA0 送信終了割り込み(INTDMA0)	使用する
			優先順位	低

A/D コンバータ				使用する
	ADC			使用する
			A/D コンバータ動作設定	使用する
			コンパレータ動作設定	許可
			分解能設定	8 ビット
			VREF(+)設定	VDD
			VREF(-)設定	VSS
			トリガ・モード設定	ソフトウエア・トリガ・モード
			動作モード設定	連続スキャン・モード
			ANI0 - ANI7 アナログ入力端子設定	ANI0 - ANI3
			変換開始チャネル設定	ANI0 - ANI3
			変換時間モード	標準 1
			変換時間	34 (1088/fCLK)(μs)
			変換結果上限/下限値設定	ADLL≦ADCRH≦ADUL で割り込み要求信号(INTAD)を発生
			上限値(ADUL)	255
			下限値(ADLL)	0
			A/D の割り込み許可(INTAD)	使用しない

[API 設定例]

r_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Enable DMA0 transfer */
    R_DMAC0_Start();

    /* Start the AD converter */
    R_ADC_Start();

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

r_cg_dmac_user.c

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

/* Start user code for pragma. Do not edit comment generated here */
#pragma address (g_adc_buf = 0x0ffe00)
/* End user code. Do not edit comment generated here */

/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_adc_buf[5][4];
volatile uint8_t g_adc_buf_cnt = 0U;
/* End user code. Do not edit comment generated here */

static void __near r_dmac0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    uint8_t i;
    uint8_t j;
    uint16_t temp;

    /* Stop the AD converter */
    R_ADC_Stop();

    /* Disable DMA0 transfer */
    R_DMAC0_Stop();

    /* Change DMA0_RAM address */
    if ((++g_adc_buf_cnt) < 4U)
    {
        DRA0 += 4U;
    }
    else
    {
        DRA0 = _FE00_DMA0_RAM_ADDRESS;
        g_adc_buf_cnt = 0U;
```

```
            /* Calculate the average */
            for (i = 0; i < 4U; i++)
            {
                temp = 0U;
                for (j = 0; j < 4U; j++)
                {
                    temp += g_adc_buf[j][i];
                }
                g_adc_buf[4][i] = temp / 4U;
            }
        }

        /* Enable DMA0 transfer */
        R_DMAC0_Start();

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

3.3.36 データ・トランスファ・コントローラ

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

表 3.38 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_DTCn_Start	データ・トランスファ・コントローラを動作可能状態にします。
R_DTCn_Stop	データ・トランスファ・コントローラを動作停止状態にします。
R_DTC_Set_PowerOff	データ・トランスファ・コントローラに対するクロック供給を停止します。
R_DTCDn_Start	データ・トランスファ・コントローラを動作可能状態にします。
R_DTCDn_Stop	データ・トランスファ・コントローラを動作停止状態にします。

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。

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

[指定形式]

```
void    R_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*n*_Start

データ・トランスファ・コントローラを動作許可状態にします。

[指定形式]

void	R_DTC*n*_Start (void);

備考 *n* は、チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_DTC*n*_Stop

データ・トランスファ・コントローラを動作停止状態にします。

[指定形式]

void	R_DTC*n*_Stop (void);

備考 *n* は、チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_DTC_Set_PowerOff

データ・トランスファ・コントローラに対するクロック供給を停止します。

備考　本 API 関数の呼び出しにより、データ・トランスファ・コントローラはリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

```
void    R_DTC_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

R_DTCD*n*_Start

データ・トランスファ・コントローラを動作許可状態にします。

[指定形式]

void	R_DTCD*n*_Start (void);

備考 *n* は、チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_DTCD*n*_Stop

データ・トランスファ・コントローラを動作停止状態にします。

[指定形式]

void	R_DTCD*n*_Stop (void);

備考 *n* は、チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

使用例

UART0 受信で DTC データ転送を開始する（4Byte の受信を 4Byte 配列 RAM に繰り返し格納）

[GUI 設定例]

データトランスファコントローラ				使用する
	DTC			使用する
		DTCBA		
			DTC ベースアドレス	0xffd00
			コントロールデータ 0 (DTCD0)	使用する(チェイン転送:使用しない; 起動要因:UART0 受信/CSI01/IIC01 転送完了または CSI01 バッファ空き)
		DTCD0		
			転送モード設定	リピートモード
			リピートモード割り込み設定	禁止
			リピートエリア設定	転送先がリピートエリア
			転送元アドレス	0xff12 固定
			転送先アドレス	0xfb00
			転送回数	4
			ブロックサイズ	1

シリアル				使用する
	SAU0			使用する
		Channel 0		
			チャネル 0	UART0(受信機能)
			データ・ビット長設定 (受信機能)	8 ビット
			データ転送方向設定 (受信機能)	LSB
			パリティ設定 (受信機能)	パリティなし
			ストップ・ビット長設定 (受信機能)	1 ビット固定です
			受信データ・レベル設定	標準
			転送レート設定 (受信機能)	9600(bps)(誤差 : +0.16% 許容最小 : -5.17% 許容最大 : +5.16%)
			受信完了割り込み設定(INTSR0)	低
			受信完了 (コールバック機能設定)	使用しない
			エラー (コールバック機能設定)	使用しない

[API 設定例]

r_main.c

```
/* Start user code for pragma. Do not edit comment generated here */
#pragma address (g_uart0_buf = 0x0ffb00)
/* End user code. Do not edit comment generated here */

/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_uart0_buf[4];
/* End user code. Do not edit comment generated here */

void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Enable DTCD0 module operation */
    R_DTCD0_Start();

    /* Start the UART0 module operation */
    R_UART0_Start();

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

3.3.37 イベントリンクコントローラ

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

表 3.39 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_Stop	イベントリンクコントローラを動作停止状態にします。

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。

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

[指定形式]

```
void    R_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_Stop

イベントリンクコントローラを動作停止状態にします。

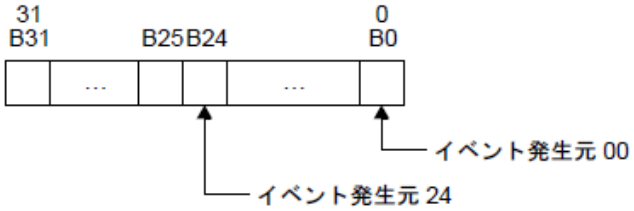
[指定形式]

void R_ELC_Stop (uint32_t *event*);

[引数]

I/O	引数	説明
I	uint32_t *event*;	停止するイベント発生元

備考 以下に、停止するイベント発生元 event の指定形式を示します。
なお、event に 0x01010101 を設定した場合、イベント発生元 00，08，16，24 のイベントリンク動作が禁止されます。



[戻り値]

なし

使用例

ELC を使用して外部割り込みエッジ検出 0 で AD 変換を開始し、変換結果を RAM に格納後 ELC 終了

[GUI 設定例]

イベントリンクコントローラ				使用する
	ELC			使用する
			AD 変換開始	使用する
			イベント発生元	外部割り込みエッジ検出 0

割り込み				使用する
	INTP			使用する
		INTP0		
			有効エッジ	立下りエッジ
			優先順位	低

A/D コンバータ				使用する
	ADC			使用する
			A/D コンバータ動作設定	使用する
			コンパレータ動作設定	許可
			分解能設定	10 ビット
			VREF(+)設定	VDD
			VREF(-)設定	VSS
			トリガ・モード設定	ハードウェア・トリガ・ノーウエイト・モード
			ハードウェア・トリガ・ノーウエイト・モード	ELC
			動作モード設定	ワンショット・セレクト・モード
			ANI0 - ANI23 アナログ入力端子設定	ANI0
			変換開始チャネル設定	ANI0
			変換時間モード	標準 1
			変換時間	38 (1216/fCLK)(μs)
			変換結果上限/下限値設定	ADLL≦ADCRH≦ADUL で割り込み要求信号(INTAD)を発生
			上限値(ADUL)	255
			下限値(ADLL)	0
			A/D の割り込み許可(INTAD)	使用する
			優先順位	低

[API 設定例]

r_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start the AD converter */
    R_ADC_Start();

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

r_cg_adc_user.c

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

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

static void __near r_adc_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop the AD converter */
    R_ADC_Stop();

    /* Return the conversion result in the buffer */
    R_ADC_Get_Result((uint16_t *)&g_adc_value);

    /* Stop the ELC event resources */
    R_ELC_Stop(0x00000001U);
    /* End user code. Do not edit comment generated here */
}
```

3.3.38 割り込み機能

以下に、コード生成ツールが割り込み機能用として出力する API 関数の一覧を示します。

表 3.40 割り込み機能用 API 関数

API 関数名	機能概要
R_INTC_Create	割り込み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_INTC_Create_UserInit	割り込み機能に関するユーザ独自の初期化処理を行います。
r_intcn_interrupt	外部マスカブル割り込み INTP*n* の発生に伴う処理を行います。
R_INTCn_Start	外部マスカブル割り込み INTP*n* の受け付けを許可します。
R_INTCn_Stop	外部マスカブル割り込み INTP*n* の受け付けを禁止します。
r_intclrn_interrupt	外部マスカブル割り込み INTPLR*n* の発生に伴う処理を行います。
R_INTCLRn_Start	外部マスカブル割り込み INTPLR*n* の受け付けを許可します。
R_INTCLRn_Stop	外部マスカブル割り込み INTPLR*n* の受け付けを禁止します。
r_intrtcicn_interrupt	外部マスカブル割り込み INTRTCIC*n* の発生に伴う処理を行います。
R_INTRTCICn_Start	外部マスカブル割り込み INTRTCIC*n* の受け付けを許可します。
R_INTRTCICn_Stop	外部マスカブル割り込み INTRTCIC*n* の受け付けを禁止します。
R_INTFO_Start	外部マスカブル割り込み INTFO の受け付けを許可します。
R_INTFO_Stop	外部マスカブル割り込み INTFO の受け付けを禁止します。
R_INTFO_ClearFlag	割り込みフラグ出力制御レジスタ（INTFOCTL1）の INTGCLR フラグをセットします。
r_intfo_interrupt	外部マスカブル割り込み INTFO の発生に伴う処理を行います。

R_INTC_Create

割り込み機能を制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_INTC_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_INTC_Create_UserInit

割り込み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_INTC_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_INTC_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_intc*n*_interrupt

外部マスカブル割り込み INTP*n* の発生に伴う処理を行います。

備考　本 API 関数は、外部マスカブル割り込み INTP*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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_intcn_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_intcn_interrupt ( void );
```

備考　*n* は、割り込み要因番号を意味します。

[引数]

なし

[戻り値]

なし

R_INTC*n*_Start

外部マスカブル割り込み INTP*n* の受け付けを許可します。

[指定形式]

void	R_INTC*n*_Start (void);

備考 *n* は、割り込み要因番号を意味します。

[引数]

なし

[戻り値]

なし

R_INTC*n*_Stop

外部マスカブル割り込み INTP*n* の受け付けを禁止します。

[指定形式]

void	R_INTC*n*_Stop (void);

備考 *n* は、割り込み要因番号を意味します。

[引数]

なし

[戻り値]

なし

r_intclr*n*_interrupt

外部マスカブル割り込み INTPLR*n* の発生に伴う処理を行います。

備考　本 API 関数は、外部マスカブル割り込み INTPLR*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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_intclrn_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_intclrn_interrupt ( void );
```

備考　*n* は、割り込み要因番号を意味します。

[引数]

なし

[戻り値]

なし

R_INTCLR*n*_Start

外部マスカブル割り込み INTPLR*n* の受け付けを許可します。

[指定形式]

void	R_INTCLR*n*_Start (void);

備考 *n* は、割り込み要因番号を意味します。

[引数]

なし

[戻り値]

なし

R_INTCLR*n*_Stop

外部マスカブル割り込み INTPLR*n* の受け付けを禁止します。

[指定形式]

void	R_INTCLR*n*_Stop (void);

備考 *n* は、割り込み要因番号を意味します。

[引数]

なし

[戻り値]

なし

r_intrtcic*n*_interrupt

外部マスカブル割り込み INTRTCIC*n* の発生に伴う処理を行います。

備考　本 API 関数は、外部マスカブル割り込み INTRTCIC*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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_intrtcicn_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_intrtcicn_interrupt ( void );
```

備考　*n* は、割り込み要因番号を意味します。

[引数]

なし

[戻り値]

なし

R_INTRTCIC*n*_Start

外部マスカブル割り込み INTRTCIC*n* の受け付けを許可します。

[指定形式]

void	R_INTRTCIC*n*_Start (void);

備考 *n* は、割り込み要因番号を意味します。

[引数]

なし

[戻り値]

なし

R_INTRTCIC*n*_Stop

外部マスカブル割り込み INTRTCIC*n* の受け付けを禁止します。

[指定形式]

void	R_INTRTCIC*n*_Stop (void);

備考 *n* は、割り込み要因番号を意味します。

[引数]

なし

[戻り値]

なし

R_INTFO_Start

外部マスカブル割り込み INTFO の受け付けを許可します。

[指定形式]

```
void        R_INTFO_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_INTFO_Stop

外部マスカブル割り込み INTFO の受け付けを禁止します。

[指定形式]

```
void        R_INTFO_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_INTFO_ClearFlag

割り込みフラグ出力制御レジスタ（INTFOCTL1）の INTFCLR フラグをセットします。

[指定形式]

```
void        R_INTFO_ClearFlag ( void );
```

[引数]

なし

[戻り値]

なし

r_intfo_interrupt

外部マスカブル割り込み INTFO の発生に伴う処理を行います。

備考　本 API 関数は、外部マスカブル割り込み INTFO 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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_intfo_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_intfo_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

使用例

立ち下がりエッジの入力回数を数える

[GUI 設定例]

割り込み				使用する
	INTP			使用する
		INTP0		
			有効エッジ	立下りエッジ
			優先順位	低

[API 設定例]

r_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Clear INTP0 interrupt flag and enable interrupt */
    R_INTC0_Start();

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

r_cg_intc_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_intc0_cnt = 0U;
/* End user code. Do not edit comment generated here */

static void __near r_intc0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Count INTP0 */
    g_intc0_cnt++;
    /* End user code. Do not edit comment generated here */
}
```

3.3.39 キー割り込み機能

以下に、コード生成ツールがキー割り込み機能用として出力する API 関数の一覧を示します。

表 3.41 キー割り込み機能用 API 関数

API 関数名	機能概要
R_KEY_Create	キー割り込み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_KEY_Create_UserInit	キー割り込み機能に関するユーザ独自の初期化処理を行います。
r_key_interrupt	キー割り込み INTKR の発生に伴う処理を行います。
R_KEY_Start	キー割り込み INTKR の受け付けを許可します。
R_KEY_Stop	キー割り込み INTKR の受け付けを禁止します。

R_KEY_Create

キー割り込み機能を制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_KEY_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_KEY_Create_UserInit

キー割り込み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_KEY_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_KEY_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_key_interrupt

キー割り込み INTKR の発生に伴う処理を行います。

備考　　本 API 関数は、キー割り込み INTKR 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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_key_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_key_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_KEY_Start

キー割り込み INTKR の受け付けを許可します。

[指定形式]

```
void        R_KEY_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_KEY_Stop

キー割り込み INTKR の受け付けを禁止します。

[指定形式]

```
void        R_KEY_Stop ( void );
```

[引数]

なし

[戻り値]

なし

使用例

8 チャネルのキーに立ち下がりエッジ入力を検出した際、キーに対応したビットフラグを立てる

[GUI 設定例]

割り込み				使用する
	KEY			使用する
			KR0	使用する
			KR1	使用する
			KR2	使用する
			KR3	使用する
			KR4	使用する
			KR5	使用する
			KR6	使用する
			KR7	使用する
			優先順位	高

[API 設定例]

r_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Clear INTKR interrupt flag and enable interrupt */
    R_KEY_Start();

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

r_cg_intc_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint8_t g_key_fix = 0x00U;
/* End user code. Do not edit comment generated here */

static void __near r_key_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    g_key_fix = ~P7;
    /* End user code. Do not edit comment generated here */
}
```

3.3.40 電圧検出回路

以下に、コード生成ツールが電圧検出回路用として出力する API 関数の一覧を示します。

表 3.42 電圧検出回路用 API 関数

API 関数名	機能概要
R_LVD_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_LVD_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_interrupt	電圧検出割り込み INTLVI の発生に伴う処理を行います。
r_lvd_vddinterrupt	VDD 端子電圧検出割り込み INTLVDVDD の発生に伴う処理を行います。
r_lvd_vbatinterrupt	VBAT 端子電圧検出割り込み INTLVDVBAT の発生に伴う処理を行います。
r_lvd_vrtcinterrupt	VRTC 端子電圧検出割り込み INTLVDVRTC の発生に伴う処理を行います。
r_lvd_exlvdinterrupt	EXLVD 端子電圧検出割り込み INTVLDEXLVD の発生に伴う処理を行います。
R_LVD_InterruptMode_Start	電圧検出動作を開始します。（割り込みモード時、および割り込み&リセット・モード時）
R_LVD_Start_VDD	VDD 端子の電圧検出機能を動作許可状態に設定します。
R_LVD_Start_VBAT	VBAT 端子の電圧検出機能を動作許可状態に設定します。
R_LVD_Start_VRTC	VRTC 端子の電圧検出機能を動作許可状態に設定します。
R_LVD_Start_EXLVD	EXLVD 端子の電圧検出機能を動作許可状態に設定します。
R_LVD_Stop_VDD	VDD 端子の電圧検出機能を動作禁止状態に設定します。
R_LVD_Stop_VBAT	VBATDD 端子の電圧検出機能を動作禁止状態に設定します。
R_LVD_Stop_VRTC	VRTC 端子の電圧検出機能を動作禁止状態に設定します。
R_LVD_Stop_EXLVD	EXLVD 端子の電圧検出機能を動作禁止状態に設定します。
R_LVI_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_LVI_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_lvi_interrupt	電圧検出割り込み INTLVI の発生に伴う処理を行います。
R_LVI_InterruptMode_Start	電圧検出動作を開始します。（割り込みモード時、および割り込み&リセット・モード時）

R_LVD_Create

電圧検出回路を制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_LVD_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_LVD_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_LVD_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_LVD_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_lvd_interrupt

電圧検出割り込み INTLVI の発生に伴う処理を行います。

備考　本 API 関数は、電圧検出割り込み INTLVI 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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_lvd_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_lvd_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_lvd_vddinterrupt

VDD 端子電圧検出割り込み INTLVDVDD の発生に伴う処理を行います。

備考　本 API 関数は、VDD 端子電圧検出割り込み INTLVDVDD 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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_lvd_vddinterrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_lvd_vddinterrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_lvd_vbatinterrupt

VBAT 端子電圧検出割り込み INTLVDVBAT の発生に伴う処理を行います。

備考　本 API 関数は、VBAT 端子電圧検出割り込み INTLVDVBAT 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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_lvd_vbatinterrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_lvd_vbatinterrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_lvd_vrtcinterrupt

VRTC 端子電圧検出割り込み INTLVDVRTC の発生に伴う処理を行います。

備考　本 API 関数は、VRTC 端子電圧検出割り込み INTLVDVRTC 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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_lvd_vrtcinterrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_lvd_vrtcinterrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_lvd_exlvdinterrupt

EXLVD 端子電圧検出割り込み INTLVDEXLVD の発生に伴う処理を行います。

備考　本 API 関数は、EXLVD 端子電圧検出割り込み INTLVDEXLVD 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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_lvd_exlvdinterrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_lvd_exlvdinterrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_LVD_InterruptMode_Start

電圧検出動作を開始します（割り込みモード時、および割り込み&リセットモード時）。

[指定形式]

```
void        R_LVD_InterruptMode_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_LVD_Start_VDD

VDD 端子の電圧検出機能を動作許可状態に設定します。

[指定形式]

```
void        R_LVD_Start_VDD ( void );
```

[引数]

なし

[戻り値]

なし

R_LVD_Start_VBAT

VBAT 端子の電圧検出機能を動作許可状態に設定します。

[指定形式]

```
void        R_LVD_Start_VBAT ( void );
```

[引数]

なし

[戻り値]

なし

R_LVD_Start_VRTC

VRTC 端子の電圧検出機能を動作許可状態に設定します。

[指定形式]

```
void        R_LVD_Start_VRTC ( void );
```

[引数]

なし

[戻り値]

なし

R_LVD_Start_EXLVD

EXLVD 端子の電圧検出機能を動作許可状態に設定します。

[指定形式]

```
void        R_LVD_Start_EXLVD ( void );
```

[引数]

なし

[戻り値]

なし

R_LVD_Stop_VDD

VDD 端子の電圧検出機能を動作禁止状態に設定します。

[指定形式]

```
void        R_LVD_Stop_VDD ( void );
```

[引数]

なし

[戻り値]

なし

R_LVD_Stop_VBAT

VBAT 端子の電圧検出機能を動作禁止状態に設定します。

[指定形式]

```
void        R_LVD_Stop_VBAT ( void );
```

[引数]

なし

[戻り値]

なし

R_LVD_Stop_VRTC

VRTC 端子の電圧検出機能を動作禁止状態に設定します。

[指定形式]

```
void        R_LVD_Stop_VRTC ( void );
```

[引数]

なし

[戻り値]

なし

R_LVD_Stop_EXLVD

EXLVD 端子の電圧検出機能を動作禁止状態に設定します。

[指定形式]

```
void        R_LVD_Stop_EXLVD ( void );
```

[引数]

なし

[戻り値]

なし

R_LVI_Create

電圧検出回路を制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_LVI_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_LVI_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_LVI_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_LVI_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_lvi_interrupt

電圧検出割り込み INTLVI の発生に伴う処理を行います。

備考　本 API 関数は、電圧検出割り込み INTLVI 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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_lvi_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_lvi_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_LVI_InterruptMode_Start

電圧検出動作を開始します（割り込みモード時、および割り込み&リセットモード時）。

[指定形式]

```
void        R_LVI_InterruptMode_Start ( void );
```

[引数]

なし

[戻り値]

なし

使用例

割り込み＆リセット・モードを使用し、リセット前に行うべき処理を割り込みで行う

[GUI 設定例]

電圧検出回路				
	LVD			使用する
			電圧検出動作設定	使用する
			動作モード設定	割込み＆リセットモード
			INTLVI 優先順位	低
			リセット発生電圧 (VLVDL)	2.75(V)
			割込み発生電圧 (VLVDH)	3.15(V)

[API 設定例]

r_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Enable the voltage detector interrupt */
    R_LVD_InterruptMode_Start();

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

r_cg_lvd_user.c

```
static void __near r_lvd_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /*** Processing to be performed before the reset ***/
    /* End user code. Do not edit comment generated here */
}
```

3.3.41 バッテリ・バックアップ機能

以下に、コード生成ツールがバッテリ・バックアップ機能用として出力する API 関数の一覧を示します。

表 3.43 バッテリ・バックアップ機能用 API 関数

API 関数名	機能概要
R_BUP_Create	バッテリ・バックアップ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_BUP_Create_UserInit	バッテリ・バックアップ機能に関するユーザ独自の初期化処理を行います。
r_bup_interrupt	電源切り替え検出割り込み INTVBAT の発生に伴う処理を行います。
R_BUP_Start	バッテリ・バックアップ機能を動作許可状態に設定します。
R_BUP_Stop	バッテリ・バックアップ機能を動作停止状態に設定します。

R_BUP_Create

バッテリ・バックアップ機能を制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_BUP_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_BUP_Create_UserInit

バッテリ・バックアップ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_BUP_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_BUP_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_bup_interrupt

電源切り替え検出割り込み INTVBAT の発生伴う処理を行います。

備考　本 API 関数は、電源切り替え検出割り込み INTVBAT 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]

CA78K0R コンパイラの場合

```
__interrupt static void      r_bup_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void      __near r_bup_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_BUP_Start

バッテリ・バックアップ機能を動作許可状態に設定します。

[指定形式]

```
void        R_BUP_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_BUP_Stop

バッテリ・バックアップ機能を動作停止状態に設定します。

[指定形式]

```
void        R_BUP_Stop ( void );
```

[引数]

なし

[戻り値]

なし

使用例

バッテリ・バックアップ・モードに入ることを知らせるフラグを立てる

[GUI 設定例]

バッテリ・バックアップ機能				使用する
	KEYBATTERYBACKUP			使用する
			バッテリ・バックアップ機能動作設定	使用する
			電源切り替え割り込み信号発生 (INTVBAT)	使用する
			電源切り替え割り込み選択	VDD → VBAT 切り替え時に割り込み発生
			優先順位	低

[API 設定例]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start battery backup module operation */
    R_BUP_Start();

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

r_cg_bup_user.c

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

static void __near r_bup_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* When entering the battery backup mode, a flag is set */
    if (VBATCMPM == 0U) {
        g_bup_f = 0U;
    }
    else
    {
        g_bup_f = 1U;
    }
    /* End user code. Do not edit comment generated here */
}
```

3.3.42 発振停止検出回路

以下に、コード生成ツールが発振停止検出回路用として出力する API 関数の一覧を示します。

表 3.44 発振停止検出回路用 API 関数

API 関数名	機能概要
R_OSDC_Create	発振停止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_OSDC_Create_UserInit	発振停止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_osdc_interrupt	発振停止検出割り込み INTOSDC の発生に伴う処理を行います。
R_OSDC_Start	発振停止検出回路を動作許可状態に設定します。
R_OSDC_Stop	発振停止検出回路を動作停止状態に設定します。
R_OSDC_Set_PowerOff	発振停止検出回路に対するクロック供給を停止します。
R_OSDC_Reset	発振停止検出回路をリセットします。

R_OSDC_Create

発振停止検出回路を制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_OSDC_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_OSDC_Create_UserInit

発振停止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_OSDC_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_OSDC_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_osdc_interrupt

発振停止検出割り込み INTOSDC の発生伴う処理を行います。

備考　本 API 関数は、発振停止検出割り込み INTOSDC 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]

CA78K0R コンパイラの場合

```
__interrupt static void　　r_osdc_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void　　__near r_osdc_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_OSDC_Start

発振停止検出回路を動作許可状態に設定します。

[指定形式]

```
void        R_OSDC_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_OSDC_Stop

発振停止検出回路を動作停止状態に設定します。

[指定形式]

```
void        R_OSDC_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_OSDC_Set_PowerOff

発振停止検出回路に対するクロック供給を停止します。

備考　本 API 関数の呼び出しにより、振停止検出回路はリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

```
void    R_OSDC_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

R_OSDC_Reset

発振停止検出回路をリセットします。

[指定形式]

```
void        R_OSDC_Reset ( void );
```

[引数]

なし

[戻り値]

なし

使用例

発振停止検出割り込みでリセットする

[GUI 設定例]

発振停止検出回路				使用する
	OSCSTOPDETECTOR			使用する
			発振停止検出回路動作設定	使用する
			発振停止判定時間	100(ms) (TYP.)(実際の値：100)
			発振停止検出割り込み信号出力(INTOSDC)	使用する
			優先順位	低

ウォッチドッグ・タイマ				
	WDT			使用する
			ウォッチドッグ・タイマ動作設定	使用する
			HALT/STOP/SNOOZE モード時の動作設定	許可
			オーバフロー時間	2^16/fIL 4369.07(ms)
			ウインドウ・オープン期間	100(%)
			オーバフロー時間の 75% + 1/2fIL 到達時にインターバル割り込みを発生する(INTWDTI)	使用する
			優先順位	低

[API 設定例]

r_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start OSDC module operation */
    R_OSDC_Start();

    while (1U)
    {
        /* Restart the watchdog timer */
        R_WDT_Restart();
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_osdc_user.c

```
static void __near r_osdc_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop OSDC module operation */
    R_OSDC_Stop();
```

```
    /* Reset */
    WDTE = 0x00;
    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

3.3.43 SPI インタフェース

以下に、コード生成ツールが SPI 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 3.45 SPI インタフェース用 API 関数

API 関数名	機能概要
R_SAIC_Create	SPI 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_SAIC_Create_UserInit	SPI 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_SAIC_Write	データの SPI 送信を開始します。
R_SAIC_Read	データの SPI 受信を開始します。
R_SPI_Create	SPI 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_SPI_Create_UserInit	SPI 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_SPI_Start	SPI 通信を待機状態にします。
R_SPI_Stop	SPI 通信を終了します。

R_SAIC_Create

SPI インタフェースを制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_SAIC_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_SAIC_Create_UserInit

SPI 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_SAIC_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_SAIC_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

R_SAIC_Write

データの SPI 送信を開始します。

[指定形式]

void R_SAIC_Write (const smartanalog_t * *p_saic_data*);

[引数]

I/O	引数	説明
I	const smartanalog_t * *p_saic_data*;	送信するデータを格納した領域へのポインタ

[戻り値]

なし

R_SAIC_Read

データの SPI 受信を開始します。

[指定形式]

void R_SAIC_Read (const smartanalog_t * *p_saic_data*, smartanalog_t * *p_saic_read_buf*);

[引数]

I/O	引数	説明
O	const smartanalog_t * *p_saic_data*;	受信したデータを格納する領域へのポインタ
O	smartanalog_t * *p_saic_read_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]

なし

R_SPI_Create

SPI インタフェースを制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_SPI_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_SPI_Create_UserInit

SPI 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_SPI_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_SPI_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

R_SPI_Start

SPI 通信を待機状態にします。

[指定形式]

```
void        R_SPI_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_SPI_Stop

SPI 通信を終了します。

[指定形式]

```
void        R_SPI_Stop ( void );
```

[引数]

なし

[戻り値]

なし

使用例

SPI 送信にて外部機器に書き込みを行い、ベリファイチェックのため書き込んだデータを読み出す
※SA-Designer と合わせてご使用ください

[GUI 設定例]

SPI インタフェース				
	SPI			使用する
			アナログ IC 動作設定	使用する
			ボー・レート	300(bps) (実際の値：300.481)

[API 設定例]

r_cg_main.c

```
/* Start user code for global. Do not edit comment generated here */
uint8_t g_flag;
smartanalog_t gp_sa_read_buf[];
extern const smartanalog_t gp_smartanalog_data[];
/* End user code. Do not edit comment generated here */

void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    while (1U)
    {
        if (1U == g_flag)
        {
            /* Write SAIC register */
            R_SAIC_Write(gp_smartanalog_data);

            /* Read SAIC register */
            R_SAIC_Read(gp_smartanalog_data, gp_sa_read_buf);

            /** read after write verify **/
        }
    }
    /* End user code. Do not edit comment generated here */
}
```

3.3.44 オペアンプ

以下に、コード生成ツールがオペアンプ用として出力する API 関数の一覧を示します。

表 3.46 オペアンプ用 API 関数

API 関数名	機能概要
R_OPAMP_Create	オペアンプを制御するうえで必要となる初期化処理を行います。
R_OPAMP_Create_UserInit	オペアンプに関するユーザ独自の初期化処理を行います。
R_OPAMP_Set_ReferenceCircuitOn	オペアンプ・リファレンス電流回路を動作許可します。
R_OPAMP_Set_ReferenceCircuitOff	オペアンプ・リファレンス電流回路を停止します。
R_OPAMPn_Start	ユニット *n* のオペアンプを動作します。
R_OPAMPn_Stop	ユニット *n* のオペアンプを停止します。
R_OPAMPn_Set_PrechargeOn	ユニット *n* のオペアンプの外部コンデンサのプリチャージを許可状態にします。
R_OPAMPn_Set_PrechargeOff	ユニット *n* のオペアンプの外部コンデンサのプリチャージを停止状態にします。

R_OPAMP_Create

オペアンプを制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_OPAMP_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_OPAMP_Create_UserInit

オペアンプ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_OPAMP_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_OPAMP_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

R_OPAMP_Set_ReferenceCircuitOn

オペアンプ・リファレンス電流回路を動作します。

[指定形式]

```
void        R_OPAMP_Set_ReferenceCircuitOn ( void );
```

[引数]

なし

[戻り値]

なし

R_OPAMP_Set_ReferenceCircuitOff

オペアンプ・リファレンス電流回路を停止します。

[指定形式]

```
void        R_OPAMP_Set_ReferenceCircuitOff ( void );
```

[引数]

なし

[戻り値]

なし

R_OPAMP*n*_Start

ユニット *n* のオペアンプを動作します。

[指定形式]

void	R_OPAMP*n*_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_OPAMP*n*_Stop

ユニット *n* のオペアンプを停止します。

[指定形式]

void	R_OPAMP*n*_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_OPAMP*n*_Set_PrechargeOn

ユニット *n* のオペアンプの外部コンデンサのプリチャージを許可状態にします。

[指定形式]

void	R_OPAMP*n*_Set_PrechargeOn (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_OPAMP*n*_Set_PrechargeOff

ユニット *n* のオペアンプの外部コンデンサのプリチャージを停止状態にします。

[指定形式]

void	R_OPAMP*n*_Set_PrechargeOff (void);

備考 *n* はユニット番号を意味します。

[引数]

なし

[戻り値]

なし

使用例

ペアンプをコンパレータの+側信号入力に使用する

[GUI 設定例]

オペアンプ				
	OPAMP			使用する
			オペアンプ動作設定	使用する
			使用オペアンプ 0	使用する
			使用オペアンプ 1	使用しない
			使用オペアンプ 2	使用しない
			使用オペアンプ 3	使用しない
			リファレンス電流回路設定	停止
			動作モード設定	ロウパワー・モード
			ELC トリガ設定	オペアンプ　ユニット 0 : オペアンプ ELC トリガ 0
オペアンプ　ユニット 1 : オペアンプ ELC トリガ 1				
オペアンプ　ユニット 2 : オペアンプ ELC トリガ 2				
オペアンプ　ユニット 3 : オペアンプ ELC トリガ 3				
		Operational Amplifier 0		使用する
			Operational amplifier 0 起動/停止トリガ制御設定	ソフトウエア・トリガ・モード

コンパレータ				
	COMP			使用する
			動作設定	コンパレータ 0 のみで使用する場合(コンパレータ入力：AMP0O)
			速度設定	低速
		Comparator0		使用する
			Comparator0 モード設定	基本
			Comparator0 エッジ設定	立上りエッジ
			Comparator0 デジタルフィルタ許可	使用しない
			Comparator0 出力許可 (VCOUT0)	使用しない
			Comparator0 コンパレータ 0 の割り込み許可 (INTCMP0)	使用する
			Comparator0 優先順位	低

[API 設定例]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start the operational amplifier 0 */
    R_OPAMP0_Start();

    /* Start the comparator 0 */
    R_COMP0_Start();
```

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

3.3.45 データ演算回路

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

表 3.47 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_interrupt	演算結果検出割り込み INTDOC の発生に伴う処理を行います。
R_DOC_SetMode	データ演算回路の動作モードを設定します。
R_DOC_WriteData	演算対象の 16 ビットのデータを設定します。
R_DOC_GetResult	データの加算結果または減算結果を取得します。
R_DOC_ClearFlag	DOC コントロールレジスタ （DOCR） の DOPCF フラグをクリアします。
R_DOC_Set_PowerOff	データ演算回路へのクロック供給を停止します。
R_DOC_Reset	データ演算回路をリセットします。

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。

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

[指定形式]

```
void    R_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_interrupt

DOC 演算結果検出割り込み INTDOC の発生に伴う処理を行います。

備考　本 API 関数は、DOC 演算結果検出割り込み INTDOC 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]

CA78K0R コンパイラの場合

```
__interrupt static void      r_doc_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void      __near r_doc_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_DOC_SetMode

データ演算回路の動作モードを設定します。

[指定形式]

```
#include      “r_cg_macrodriver.h”
MD_STATUS      R_DOC_SetMode ( doc_mode_t mode, unit16_t value );
```

[引数]

I/O	引数	説明
I	doc_mode_t *mode*;	データ演算回路の動作モード ADDITION ：データ加算モード SUBTRACTION ：データ減算モード COMPARE_MATCH ：データ比較モード （結果の検出条件：一致を検出） COMPARE_MISMATCH ：データ比較モード （結果の検出条件：不一致を検出）
I	uint16_t *value*;	データ加算モードおよびデータ減算モードの場合：演算結果 データ比較モードの場合：基準となるデータ

[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_DOC_WriteData

演算対象の 16 ビットのデータを設定します。

[指定形式]

void R_DOC_WriteData (unit16_t *data*);

[引数]

I/O	引数	説明
I	unit16_t *data*;	演算対象の 16 ビットデータ

[戻り値]

なし

R_DOC_GetResult

データの加算結果または減算結果を取得します。

[指定形式]

```
void        R_DOC_GetResult ( unit16_t*const data );
```

[引数]

I/O	引数	説明
O	unit16_t*const *data*;	演算結果を格納した領域へのポインタ

[戻り値]

なし

R_DOC_ClearFlag

DOC コントロールレジスタ （DOCR） の DOPCF 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

R_DOC_Set_PowerOff

データ演算回路へのクロック供給を停止します。

[指定形式]

```
void        R_DOC_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

R_DOC_Reset

データ演算回路をリセットします。

[指定形式]

```
void        R_DOC_Reset ( void );
```

[引数]

なし

[戻り値]

なし

使用例

データ加算モードで配列データを加算し、“FFFFh”より大きくなったとき割り込みで加算結果を取得する

その後データ比較不一致モードに変更し、配列データに"000h"以外を検出した場合に割り込みを発生させる

[GUI 設定例]

データ演算回路				
	DOC			使用する
			DOC 設定	使用する
			動作モード	データ加算モード
			比較基準値/加減演算結果初期値	0xFFFF
			データ演算回路割り込み(INTDOC)有効	使用する
			INTDOC 優先順位	低

[API 設定例]

r_cg_main.c

```
/* Start user code for global. Do not edit comment generated here */
extern volatile uint16_t data[16];
volatile uint8_t cnt;
/* End user code. Do not edit comment generated here */

void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    while (1U)
    {
        for (cnt = 0; cnt < 16U; cnt++)
        {
            /* Write new data to compare */
            R_DOC_WriteData(data[cnt]);
        }
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_doc_user.c

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

static void __near r_doc_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Get result */
    R_DOC_GetResult((uint16_t *)&result);
```

```
    /* Configure the operation mode of DOC */
    R_DOC_SetMode(COMPARE_MISMATCH, 0x0000);

    /* Clear DOPCF flag */
    R_DOC_ClearFlag();
    /* End user code. Do not edit comment generated here */
}
```

3.3.46 32 ビット積和演算回路

以下に、コード生成ツールが 32 ビット積和演算回路用として出力する API 関数の一覧を示します。

表 3.48 32 ビット積和演算回路用 API 関数

API 関数名	機能概要
R_MAC32Bit_Create	32 ビット積和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_MAC32Bit_Create_UserInit	32 ビット積和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_mac32bit_interrupt_flow	32 ビット積和演算オーバーフロー / アンダーフロー割り込み INTMACLOF の発生に伴う処理を行います。
R_MAC32Bit_Reset	32 ビット積和演算回路をリセットします。
R_MAC32Bit_Set_PowerOff	32 ビット積和演算回路に対するクロック供給を停止します。
R_MAC32Bit_MULUnsigned	符号なし乗算を行います。
R_MAC32Bit_MULSigned	符号あり乗算を行います。
R_MAC32Bit_MACUnsigned	符号なし積和演算を行います。
R_MAC32Bit_MACSigned	符号あり積和演算を行います。

R_MAC32Bit_Create

32 ビット積和演算回路を制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void        R_MAC32Bit_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_MAC32Bit_Create_UserInit

32 ビット積和演算回路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_MAC32Bit_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_MAC32Bit_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_mac32bit_interrupt_flow

32 ビット積和演算オーバーフロー / アンダーフロー割り込み INTMACLOF の発生に伴う処理を行います。

備考 本 API 関数は、32 ビット積和演算オーバーフロー / アンダーフロー割り込み INTMACLOF 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]

CA78K0R コンパイラの場合

```
__interrupt static void      r_mac32bit_interrupt_flow ( void );
```

CC-RL コンパイラの場合

```
static void      __near r_mac32bit_interrupt_flow ( void );
```

[引数]

なし

[戻り値]

なし

R_MAC32Bit_Reset

32 ビット積和演算回路をリセットします。

[指定形式]

```
void        R_MAC32Bit_Reset ( void );
```

[引数]

なし

[戻り値]

なし

R_MAC32Bit_Set_PowerOff

32 ビット積和演算回路に対するクロック供給を停止します。

備考　本 API 関数の呼び出しにより、32 ビット積和演算回路はリセット状態へと移行します。このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

```
void    R_MAC32Bit_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

R_MAC32Bit_MULUnsigned

符号なし乗算を行います。

[指定形式]

```
void        R_MAC32Bit_MULUnsigned ( uint32_t data_a, uint32_t data_b, mac32bit_uint64_t
* buffer_64bit );
```

[引数]

I/O	引数	説明
I	uint32_t *data_a*;	被乗数値
I	uint32_t *data_b*;	乗数値
O	mac32bit_uint64_t * *buffer_64bit*;	演算結果

備考　　以下に、演算結果 mac32bit_uint64_t の構成を示します。

```
typedef struct
{
        uint16_t low_low;
        uint16_t low_high;
        uint16_t high_low;
        uint16_t high_high;
} mac32bit_uint64_t;
```

[戻り値]

なし

R_MAC32Bit_MULSigned

符号あり乗算を行います。

[指定形式]

```
void R_MAC32Bit_MULSigned ( int32_t data_a, int32_t data_b, mac32bit_int64_t
* buffer_64bit );
```

[引数]

I/O	引数	説明
I	int32_t *data_a*;	被乗数値
I	int32_t *data_b*;	乗数値
O	mac32bit_int64_t * *buffer_64bit*;	演算結果

備考 以下に、演算結果 mac32bit_int64_t の構成を示します。

```
typedef struct
{
        int16_t low_low;
        int16_t low_high;
        int16_t high_low;
        int16_t high_high;
} mac32bit_int64_t;
```

[戻り値]

なし

R_MAC32Bit_MACUnsigned

符号なし積和演算を行います。

[指定形式]

```
void R_MAC32Bit_MACUnsigned ( uint32_t data_a, uint32_t data_b, mac32bit_uint64_t
* buffer_64bit );
```

[引数]

I/O	引数	説明
I	uint32_t *data_a*;	被乗数値
I	uint32_t *data_b*;	乗数値
O	mac32bit_uint64_t * *buffer_64bit*;	累計初期値／演算結果

備考 累計初期値／演算結果 mac32bit_uint64_t についての詳細は、R_MAC32Bit_MULUnsigned を参照してください。

[戻り値]

なし

R_MAC32Bit_MACSigned

符号あり積和演算を行います。

[指定形式]

```
void        R_MAC32Bit_MACSigned ( int32_t data_a, int32_t data_b, mac32bit_int64_t
* buffer_64bit );
```

[引数]

I/O	引数	説明
I	int32_t *data_a*;	被乗数値
I	int32_t *data_b*;	乗数値
O	mac32bit_int64_t * *buffer_64bit*;	累計初期値／演算結果

備考　累計初期値／演算結果 mac32bit_int64_t についての詳細は、R_MAC32Bit_MULSigned を参照してください。

[戻り値]

なし

使用例

1ch の 10bit AD 変換値の 4 回平均値を算出

[GUI 設定例]

32 ビット積和演算回路動作設定				
	MAC32bit			使用する
			32 ビット積和演算回路動作設定	使用する
			固定小数点モード設定	無効
			乗算累積オーバーフロー/アンダーフローを有効にします（INTMACLOF）	使用しない

A/D コンバータ				
	ADC			使用する
			A/D コンバータ動作設定	使用する
			コンパレータ動作設定	停止
			分解能設定	10 ビット
			VREF(+)設定	VDD
			VREF(-)設定	VSS
			トリガ・モード設定	ソフトウエア・トリガ・モード
			動作モード設定	ワンショット・セレクト・モード
			ANI0 - ANI5 アナログ入力端子設定	ANI0
			変換開始チャネル設定	ANI0
			変換時間モード	標準 1
			変換時間	608/fCLK 25.3333(μs)
			変換結果上限/下限値設定	ADLL≦ADCRH≦ADUL で割り込み要求信号(INTAD)を発生
			上限値(ADUL)	255
			下限値(ADLL)	0
			A/D の割り込み許可(INTAD)	使用しない

[API 設定例]

r_cg_main.c

```
/* Start user code for global. Do not edit comment generated here */
volatile mac32bit_uint64_t g_mac32bit_buf;
volatile uint16_t g_adc_fix;
volatile uint16_t g_buffer;
volatile uint8_t g_cnt;
/* End user code. Do not edit comment generated here */

void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start the AD converter */
    R_ADC_Start();
```

```
    while (1U)
    {
        while (0U == ADIF)
        {
            ;
        }
        ADIF = 0U;

        /* Return the conversion result in the buffer */
        R_ADC_Get_Result((uint16_t *)&g_buffer);

        /* Caculate unsigned values in multiply-accumulation mode */
        R_MAC32Bit_MACUnsigned(1U, g_buffer, (mac32bit_uint64_t *)&g_mac32bit_buf);

        if ((++g_cnt) >= 4U)
        {
            g_cnt = 0U;
            g_adc_fix = (g_mac32bit_buf.low_low >> 4U);
            g_mac32bit_buf.low_low     = 0U;
            g_mac32bit_buf.low_high    = 0U;
            g_mac32bit_buf.high_low    = 0U;
            g_mac32bit_buf.high_high = 0U;
        }
    }
    /* End user code. Do not edit comment generated here */
}
```

3.3.47 12 ビット A/D コンバータ

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 3.49 12 ビット A/D コンバータ用 API 関数

API 関数名	機能概要
R_12ADC_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_12ADC_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_12adc_interrupt	A/D 変換終了割り込み INTAD の発生に伴う処理を行います。
R_12ADC_Start	A/D 変換を開始します。
R_12ADC_Stop	A/D 変換を終了します。
R_12ADC_Get_ValueResult	A/D 変換結果（12 ビット）を読み出します。
R_12ADC_Set_ADChannel	A/D 変換するアナログ電圧の入力端子を設定します。
R_12ADC_TemperatureSensorOutput_On	12 ビット A/D コンバータの温度センサー回路を動作します。
R_12ADC_TemperatureSensorOutput_Off	12 ビット A/D コンバータの温度センサー回路を停止します。
R_12ADC_InternalReferenceVoltage_On	12 ビット A/D コンバータのリファレンス電圧回路を動作します。
R_12ADC_InternalReferenceVoltage_Off	12 ビット A/D コンバータのリファレンス電圧回路を停止します。
R_12ADC_Set_PowerOff	12 ビット A/D コンバータに対するクロック供給を停止します。

R_12ADC_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。

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

[指定形式]

```
void    R_12ADC_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_12ADC_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_12ADC_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_12ADC_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_12adc_interrupt

A/D 変換終了割り込み INTAD の発生伴う処理を行います。

備考　本 API 関数は、A/D 変換終了割り込み INTAD 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]

CA78K0R コンパイラの場合

```
__interrupt static void      r_12adc_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void      __near r_12adc_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_12ADC_Start

A/D 変換を開始します。

備考 電圧コンパレータが動作停止状態から動作許可状態へと移行した際、約 1μ 秒の安定時間を必要とします。
したがって、R_12ADC_Create と本 API 関数の間には、約 1μ 秒の時間を空ける必要があります。

[指定形式]

```
void    R_12ADC_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_12ADC_Stop

A/D 変換を終了します。

備考　電圧コンパレータは、本 API 関数の処理完了後も動作を継続しています。
したがって、電圧コンパレータの動作を停止する場合は、本 API 関数の処理完了後、R_12ADC_Set_PowerOff を呼び出す必要があります。

[指定形式]

```
void    R_12ADC_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_12ADC_Get_ValueResult

A/D 変換結果（12 ビット）を読み出します。

[指定形式]

```
#include      "r_cg_macrodriver.h"
MD_STATUS      R_12ADC_Get_ValueResult ( ad_channel_t channel, uint16_t * const buffer );
```

[引数]

I/O	引数	説明
I	ad_channel_t *channel*;	チャネル番号
O	uint16_t * const *buffer*;	読み出した A/D 変換結果を格納する領域へのポインタ

[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_12ADC_Set_ADChannel

A/D 変換するアナログ電圧の入力端子を設定します。

備考 引数 *regester*, *data* に指定された値は、A/D チャネル選択レジスタ A0（ADANSA0）または A/D 変換拡張入力コントロールレジスタ（ADEXICR）に設定されます。

[指定形式]

```
#include      "r_cg_macrodriver.h"
MD_STATUS     R_12ADC_Set_ADChannel ( ad_sel_regester_t regester, uint16_t data );
```

[引数]

I/O	引数	説明
I	ad_sel_regester_t *regester*;	設定するレジスタ SEL_ADANSA0 ：A/D チャネル選択レジスタ A0（ADANSA0） SEL_ADEXICR ：A/D 変換拡張入力コントロールレジスタ（ADEXICR）
I	uint16_t *data*;	制御レジスタに設定する値

[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_12ADC_TemperatureSensorOutput_On

12 ビット A/D コンバータの温度センサー回路を動作します。

[指定形式]

```
void        R_12ADC_TemperatureSensorOutput_On ( void );
```

[引数]

なし

[戻り値]

なし

R_12ADC_TemperatureSensorOutput_Off

12 ビット A/D コンバータの温度センサー回路を停止します。

[指定形式]

```
void        R_12ADC_TemperatureSensorOutput_Off ( void );
```

[引数]

なし

[戻り値]

なし

R_12ADC_InternalReferenceVoltage_On

12 ビット A/D コンバータのリファレンス電圧回路を動作します。

[指定形式]

```
void        R_12ADC_InternalReferenceVoltage_On ( void );
```

[引数]

なし

[戻り値]

なし

R_12ADC_InternalReferenceVoltage_Off

12 ビット A/D コンバータのリファレンス電圧回路を停止します。

[指定形式]

```
void        R_12ADC_InternalReferenceVoltage_Off ( void );
```

[引数]

なし

[戻り値]

なし

R_12ADC_Set_PowerOff

12 ビット A/D コンバータに対するクロック供給を停止します。

備考　本 API 関数の呼び出しにより、12 ビット A/D コンバータはリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

```
void    R_12ADC_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

使用例

2 端子の AD 変換結果を取得する

[GUI 設定例]

12 ビット A/D コンバータ				
	ADC			使用する
			A/D コンバータ動作設定	使用する
			A/D 変換クロック設定	PCLK
			A/D 変換モード設定	高速変換
			VREF（+）設定	AVDD
			VREF（-）設定	AVSS
			動作モード設定	シングルスキャンモード
			変換開始トリガ設定	ソフトウェアトリガ
			アナログ入力チャンネル設定	
			ANI00	使用する
			ANI00 加算/平均機能	使用しない
			ANI01	使用する
			ANI01 加算/平均機能	使用しない
			データレジスタ設定	
			AD 変換値加算回数	1 回変換
			データ配置	右詰め
			自動クリア	自動クリアを禁止
			ANI00 入力サンプリング時間	3.667(μs) (実際の値：3.667)
			ANI01 入力サンプリング時間	3.667(μs) (実際の値：3.667)
			A/D 変換値カウント設定	加算モード
			割り込み設定	AD 変換終了割り込みを有効にします（INTAD）
			優先順位	レベル 3(低優先順位)

[API 設定例]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start AD converter */
    R_12ADC_Start();

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

r_cg_12adc_user.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint16_t g_12adc_ch000_value;
```

```
volatile uint16_t g_12adc_ch001_value;
/* End user code. Do not edit comment generated here */

static void __near r_12adc_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop AD converter */
    R_12ADC_Stop();

    /* Get AD converter result */
    R_12ADC_Get_ValueResult(ADCHANNEL0, (uint16_t *)&g_12adc_ch000_value);
    R_12ADC_Get_ValueResult(ADCHANNEL1, (uint16_t *)&g_12adc_ch001_value);
    /* End user code. Do not edit comment generated here */
}
```

3.3.48 12 ビット D/A コンバータ

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 3.50 12 ビット D/A コンバータ用 API 関数

API 関数名	機能概要
R_12DA_Create	12 ビット 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_12DA_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_12DAn_Start	D/A 変換を開始します。
R_12DAn_Stop	D/A 変換を終了します。
R_12DAn_Set_ConversionValue	ANO*n* 端子に出力するアナログ電圧値を設定します。
R_12DA_Set_PowerOff	D/A コンバータに対するクロック供給を停止します。

R_12DA_Create

12 ビット D/A コンバータを制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_12DA_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_12DA_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_12DA_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_12DA_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

R_12DA*n*_Start

D/A 変換を開始します。

[指定形式]

void	R_12DA*n*_Start (void);

備考 *n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_12DA*n*_Stop

D/A 変換を終了します。

[指定形式]

void	R_12DA*n*_Stop (void);

備考 *n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_12DA_Set_PowerOff

12 ビット D/A コンバータに対するクロック供給を停止します。

備考　本 API 関数の呼び出しにより、12 ビット D/A コンバータはリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

```
void    R_12DA_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

R_12DA*n*_Set_ConversionValue

ANO*n* 端子に出力するアナログ電圧値を設定します。

[指定形式]

void R_12DA*n*_Set_ConversionValue (uint16_t *reg_value*);

備考 *n* はチャネル番号を意味します。

[引数]

I/O	引数	説明
I	uint16_t *reg_value*;	D/A 変換値

[戻り値]

なし

使用例

0x00 から開始した変換出力を一定周期毎に 0x10 ずつ上げて行き、0x0FFF になると変換停止

[GUI 設定例]

12 ビット D/A コンバータ				
	DA			使用する
			D/A コンバータ動作設定	使用する
			基準電圧	AVDD/AVSS
			データ形式	右詰め
			DA0 を使用	Used
			DA1 を使用	Unused
			D/A A/D 同期設定	使用しない

タイマ・アレイ・ユニット				
	TAU0			TAU0
		Channel 0		

[API 設定例]

r_cg_main.c

```
/* Start user code for global. Do not edit comment generated here */
volatile uint16_t g_12da0_value;
/* End user code. Do not edit comment generated here */

void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start TAU0 channel 0 counter */
    R_TAU0_Channel0_Start();

    /* Set the DA0 converter value */
    g_12da0_value = 0x0000U;
    R_12DA0_Set_ConversionValue((uint8_t)g_12da0_value);

    /* Enable the DA0 converter */
    R_12DA0_Start();

    while (1U)
    {
        while (TMIF00 == 0U){
        }
        TMIF00 = 0U;

        g_12da0_value += 0x0010U;
        if (g_12da0_value <= 0x0FFFU)
```

```
        {
            /* Set the DA0 converter value */
            R_12DA0_Set_ConversionValue((uint8_t)g_12da0_value);
        }
        else
        {
            /* Stop the DA0 converter */
            R_12DA0_Stop();
        }
    }
    /* End user code. Do not edit comment generated here */
}
```

3.3.49 オペアンプ&アナログスイッチ

以下に、コード生成ツールがオペアンプ&アナログスイッチ用として出力する API 関数の一覧を示します。

表 3.51 オペアンプ&アナログスイッチ用 API 関数

API 関数名	機能概要
R_AMPANSW_Create	オペアンプ&アナログスイッチを制御するうえで必要となる初期化処理を行います。
R_AMPANSW_Create_UserInit	オペアンプ&アナログスイッチに関するユーザ独自の初期化処理を行います。
R_OPAMPm_Set_ReferenceCircuitOn	ユニット *m* のオペアンプ・リファレンス電流回路を動作許可します。
R_OPAMPm_Set_ReferenceCircuitOff	ユニット *m* のオペアンプ・リファレンス電流回路を停止します。
R_OPAMPm_Start	ユニット *m* のオペアンプを動作します。
R_OPAMPm_Stop	ユニット *m* のオペアンプを停止します。
R_ANSW_ChargePumpm_On	ユニット *m* のアナログスイッチ回路を動作します。
R_ANSW_ChargePumpm_Off	ユニット *m* のアナログスイッチ回路を停止します。

R_AMPANSW_Create

オペアンプ&アナログスイッチを制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void        R_AMPANSW_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_AMPANSW_Create_UserInit

オペアンプ&アナログスイッチ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_AMPANSW_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_AMPANSW_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

R_OPAMP*m*_Set_ReferenceCircuitOn

ユニット *m* のオペアンプ・リファレンス電流回路を動作します。

[指定形式]

void	R_OPAMP*m*_Set_ReferenceCircuitOn (void);

備考 *m* はユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_OPAMP*m*_Set_ReferenceCircuitOff

ユニット *m* のオペアンプ・リファレンス電流回路を停止します。

[指定形式]

void	R_OPAMP*m*_Set_ReferenceCircuitOff (void);

備考 *m* はユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_OPAMP*m*_Start

ユニット *m* のオペアンプを動作します。

[指定形式]

void	R_OPAMP*m*_Start (void);

備考 *m* はユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_OPAMP*m*_Stop

ユニット *m* のオペアンプを停止します。

[指定形式]

void	R_OPAMP*m*_Stop (void);

備考 *m* はユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_ANSW_ChargePump*m*_On

ユニット *m* のアナログスイッチ回路を動作します。

[指定形式]

void	R_ANSW_ChargePump*m*_On (void);

備考 *m* はユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_ANSW_ChargePump*m*_Off

ユニット *m* のアナログスイッチ回路を停止します。

[指定形式]

void	R_ANSW_ChargePump*m*_Off (void);

備考 *m* はユニット番号を意味します。

[引数]

なし

[戻り値]

なし

使用例

アナログスイッチを使用してオペアンプ出力を行う

[GUI 設定例]

オペアンプ&アナログスイッチ				
	AMPANSW			使用する
		OPAMP		使用する
			オペアンプ動作の設定	使用する
			オペアンプ 0 を使用します	使用する
			オペアンプ 1 を使用します	使用する
			オペアンプ 2 を使用します	使用する
			OPAMP0/OPAMP1 基準電流回路の設定	停止
			OPAMP0/OPAMP1 動作モード設定	ロウパワー・モード
			OPAMP2 動作モード設定	ロウパワー・モード
			ELC トリガ設定	オペアンプ　ユニット 0 : オペアンプ ELC トリガ 0
オペアンプ　ユニット 1 : オペアンプ ELC トリガ 1				
オペアンプ　ユニット 2 : オペアンプ ELC トリガ 2				
			オペアンプ 0 起動/停止トリガ制御設定	ソフトウエア・トリガ・モード
		ANSW		使用する
			アナログスイッチ動作設定	使用する
			アナログマルチプレクサ MUX00 を有効にします	使用する
			アナログマルチプレクサ MUX01 を有効にします	使用する
			アナログマルチプレクサ MUX02 を有効にします	使用する
			アナログマルチプレクサ MUX03 を有効にします	使用する
			アナログマルチプレクサ MUX10 を有効にします	使用しない
			アナログマルチプレクサ MUX11 を有効にします	使用しない
			アナログマルチプレクサ MUX12 を有効にします	使用しない
			アナログマルチプレクサ MUX13 を有効にします	使用しない
			低抵抗スイッチ 0 を有効にします	使用しない
			低抵抗スイッチ 1 を有効にします	使用しない
			低抵抗スイッチ 2 を有効にします	使用しない

[API 設定例]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start charge pump 0 */
    R_ANSW_ChargePump0_On();

    /* Start the operational amplifier 0 */
    R_OPAMP0_Start();

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

3.3.50 ボルテージ・リファレンス

以下に、コード生成ツールがボルテージ・リファレンス用として出力する API 関数の一覧を示します。

表 3.52 ボルテージ・リファレンス用 API 関数

API 関数名	機能概要
R_VR_Create	ボルテージ・リファレン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_VR_Create_UserInit	ボルテージ・リファレンスに関するユーザ独自の初期化処理を行います。
R_VR_Start	VDD 端子の電圧検出機能を動作許可状態に設定します。
R_VR_Stop	VDD 端子の電圧検出機能を動作禁止状態に設定します。

R_VR_Create

ボルテージ・リファレンスを制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_VR_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_VR_Create_UserInit

ボルテージ・リファレン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_VR_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_VR_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

R_VR_Start

ボルテージ・リファレンスを動作します。

[指定形式]

```
void        R_VR_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_VR_Stop

ボルテージ・リファレンスを停止します。

[指定形式]

```
void        R_VR_Stop ( void );
```

[引数]

なし

[戻り値]

なし

使用例

A/D コンバータに、生成した基準電圧を供給する

[GUI 設定例]

ボルテージ・リファレンス				
	VR			使用する
			ボルテージ･リファレンスの動作設定	使用する
			1/2AVDD 電圧出力の動作設定	チャネル 0、1 の D/A 変換を一括許可
			VREFOUT 端子出力レベル選択	右詰め

12 ビット A/D コンバータ				
	ADC			使用する
			A/D コンバータ動作設定	使用する
			A/D 変換クロック設定	PCLK
			A/D 変換モード設定	高速変換
			VREF（+）設定	AVREFP/VREFOUT
			VREF（-）設定	AVSS
			動作モード設定	シングルスキャンモード
			変換開始トリガ設定	ソフトウェアトリガ
			アナログ入力チャンネル設定	
			ANI00	使用する
			ANI00 加算/平均機能	使用しない
			データレジスタ設定	
			AD 変換値加算回数	1 回変換
			データ配置	右詰め
			自動クリア	自動クリアを禁止
			ANI00 入力サンプリング時間	3.667(μs) (実際の値：3.667)
			A/D 変換値カウント設定	加算モード
			割り込み設定	AD 変換終了割り込みを有効にします（INTAD）
			優先順位	レベル 3(低優先順位)
			データレジスタ設定	

[API 設定例]

r_cg_main.c

```
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    /* Start VR module operation */
    R_VR_Start();

    /* Start AD converter */
    R_12ADC_Start();

    while (1U)
```

```
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

r_cg_12adc_user.c

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

/* Start user code for global. Do not edit comment generated here */
volatile uint16_t g_12adc_ch000_value;
/* End user code. Do not edit comment generated here */

static void __near r_12adc_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Stop VR module operation */
    R_VR_Stop();

    /* Stop AD converter */
    R_12ADC_Stop();

    /* Get AD converter result */
    R_12ADC_Get_ValueResult(ADCHANNEL0, (uint16_t *)&g_12adc_ch000_value);
    /* End user code. Do not edit comment generated here */
}
```

3.3.51 サンプリング出力タイマ／ディテクタ

以下に、コード生成ツールがサンプリング出力タイマ／ディテクタ用として出力する API 関数の一覧を示します。

表 3.53 サンプリング出力タイマ／ディテクタ用 API 関数

API 関数名	機能概要
R_SMOTD_Create	サンプリング出力タイマ／ディテク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_SMOTD_Create_UserInit	サンプリング出力タイマ／ディテクタに関するユーザ独自の初期化処理を行います。
r_smotd_counterA_interrupt	サンプリング出力タイマインターバル割り込み（INTSMOTA）の発生に伴う処理を行います。
r_smotd_counterB_interrupt	サンプリング出力タイマインターバル割り込み（INTSMOTB）の発生に伴う処理を行います。
r_smotd_smpn_interrupt	サンプリング・ディテクタ検出割り込みの発生に伴う処理を行います。
R_SMOTD_Start	サンプリング出力タイマ／ディテクタの動作を開始します。
R_SMOTD_Stop	サンプリング出力タイマ／ディテクタの動作を終了します。
R_SMOTD_Set_PowerOff	サンプリング出力タイマ／ディテクタに対するクロック供給を指定します。

R_SMOTD_Create

サンプリング出力タイマ／ディテクタを制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void        R_SMOTD_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_SMOTD_Create_UserInit

サンプリング出力タイマ／ディテク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_SMOTD_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_SMOTD_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_smotd_counterA_interrupt

サンプリング出力タイマインターバル割り込み（INTSMOTA） の発生伴う処理を行います。

備考　　本 API 関数は、サンプリング出力タイマインターバル割り込み（INTSMOTA 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]

CA78K0R コンパイラの場合

```
__interrupt static void      r_smotd_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void      __near r_smotd_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_smotd_counterB_interrupt

サンプリング出力タイマインターバル割り込み（INTSMOTB）の発生伴う処理を行います。

備考　本 API 関数は、サンプリング出力タイマインターバル割り込み（INTSMOTB）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]

CA78K0R コンパイラの場合

```
__interrupt static void      r_smotd_counterB_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void      __near r_smotd_counterB_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

r_smotd_smp*n*_interrupt

サンプリング・ディテクタ検出割り込みの発生伴う処理を行います。

[指定形式]

CA78K0R コンパイラの場合

```
__interrupt static void      r_smotd_smpn_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void      __near r_smotd_smpn_interrupt ( void );
```

備考　　*n* は、入力端子（SMP0-SMP5）の番号を意味します。

[引数]

なし

[戻り値]

なし

R_SMOTD_Start

サンプリング出力タイマ／ディテクタの動作を開始します。

[指定形式]

```
void        R_SMOTD_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_SMOTD_Stop

サンプリング出力タイマ／ディテクタの動作を停止します。

[指定形式]

```
void        R_SMOTD_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_SMOTD_Set_PowerOff

サンプリング出力タイマ／ディテクタに対するクロック供給を停止します。

備考　本 API 関数の呼び出しにより、サンプリング出力タイマ／ディテクタはリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

```
void    R_SMOTD_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

3.3.52 外部サンプリング

以下に、コード生成ツールがボルテージ・リファレンス用として出力する API 関数の一覧を示します。

表 3.54 ボルテージ・リファレンス用 API 関数

API 関数名	機能概要
R_EXSD_Create	外部サンプリングを制御するうえで必要となる初期化処理を行います。
R_EXSD_Create_UserInit	外部サンプリングに関するユーザ独自の初期化処理を行います。
r_exsd_interrupt	外部サンプリングエッジ検出割り込みの発生に伴う処理を行います。
R_EXSD_Start	外部サンプリングの動作を開始します。
R_EXSD_Stop	外部サンプリングの動作を終了します。
R_EXSD_Set_PowerOff	外部サンプリングに対するクロック供給を停止します。

R_EXSD_Create

外部サンプリングを制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_EXSD_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_EXSD_Create_UserInit

外部サンプリング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_EXSD_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_EXSD_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_exsd_interrupt

外部サンプリングエッジ検出割り込みの発生伴う処理を行います。

備考　本 API 関数は、外部サンプリングエッジ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]

CA78K0R コンパイラの場合

```
__interrupt static void      r_exsd_interrupt ( void );
```

CC-RL コンパイラの場合

```
static void      __near r_exsd_interrupt ( void );
```

[引数]

なし

[戻り値]

なし

R_EXSD_Start

外部サンプリングの動作を開始します。

[指定形式]

```
void        R_EXSD_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_EXSD_Stop

外部サンプリングの動作を停止します。

[指定形式]

```
void        R_EXSD_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_EXSD_Set_PowerOff

外部サンプリングに対するクロック供給を停止します。

備考　本 API 関数の呼び出しにより、外部サンプリングはリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

```
void    R_EXSD_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

3.3.53 シリアル・インタフェース UARTMG

以下に、コード生成ツールがシリアル・インタフェース UARTMG 用として出力する API 関数の一覧を示します。

表 3.55 シリアル・インタフェース UARTMG 用 API 関数

API 関数名	機能概要
R_UARTMGn_Create	シリアル・インタフェース UARTMG を制御するうえで必要となる初期化処理を行います。
R_UARTMGn_Create_UserInit	シリアル・インタフェース UARTMG に関するユーザ独自の初期化処理を行います。
r_uartmgn_interrupt_send	UARTMG 送信転送完了／送信バッファ空き割り込み INTSTMG*n* の発生に伴う処理を行います。
r_uartmgn_interrupt_receive	UARTMG 受信転送完了割り込み INTSRMG*n* の発生に伴う処理を行います。
r_uartmgn_interrupt_error	UARTMG 受信転送完了通信エラー発生割り込み INTSREMG*n* の発生に伴う処理を行います。
R_UARTMGn_Start	UART 通信を待機状態にします。
R_UARTMGn_Stop	UART 通信を終了します。
R_UARTFn_Set_PowerOff	シリアル・インタフェース UARTMG に対するクロック供給を停止します。
R_UARTMGn_Send	データの UART 送信を開始します。
R_UARTMGn_Receive	データの UART 受信を開始します。
r_uartmgn_callback_sendend	UARTMG 送信完了／送信バッファ空き割り込み INTSTMG*n* の発生に伴う処理を行います。
r_uartmgn_callback_receiveend	UARTMG 受信完了割り込み INTSRMG*n* の発生に伴う処理を行います。
r_uartmgn_callback_error	UARTMG 受信エラー割り込み INTSRMG*n*／INTSREMG*n* の発生に伴う処理を行います。
r_uartmgn_callback_softwareoverrun	オーバラン・エラーの検出に伴う処理を行います。

R_UARTMG*n*_Create

シリアル・インタフェース UARTMG を制御するうえで必要となる初期化処理を行います。

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

[指定形式]

void	R_UARTMG*n*_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_UARTMG*n*_Create_UserInit

シリアル・インタフェース UARTMG 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_UARTMGn_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_UARTMG*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_uartmg*n*_interrupt_send

UARTMG 送信転送完了／送信バッファ空き割り込み INTSTMG*n* の発生に伴う処理を行います。

備考　本 API 関数は、UARTMG 送信転送完了／送信バッファ空き割り込み INTSTMG*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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_uartmg*n*_interrupt_send ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_uartmgf*n*_interrupt_send ( 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_uartmg*n*_interrupt_receive

UARTMG 受信転送完了割り込み INTSRMG*n* の発生に伴う処理を行います。

備考 本 API 関数は、UARTMG 受信転送完了割り込み INTSRMG*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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_uartmgn_interrupt_receive ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_uartmgn_interrupt_receive ( 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_uartmg*n*_interrupt_error

UART 受信転送完了通信エラー発生割り込み INTSREMG*n* の発生に伴う処理を行います。

備考　本 API 関数は、UARTMG 受信転送完了通信エラー発生割り込み INTSREMG*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]

CA78K0R コンパイラの場合

```
__interrupt static void     r_uartmgn_interrupt_error ( void );
```

CC-RL コンパイラの場合

```
static void     __near r_uartmgn_interrupt_error ( 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_UARTMG*n*_Start

UART 通信を待機状態にします。

[指定形式]

void	R_UARTMG*n*_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_UARTMG*n*_Stop

UART 通信を終了します。

[指定形式]

void	R_UARTMG*n*_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_UARTMG*n*_Set_PowerOff

シリアル・インタフェース UARTMG に対するクロック供給を停止します。

備考 本 API 関数の呼び出しにより、シリアル・インタフェース UARTMG はリセット状態へと移行します。
このため、本 API 関数の呼び出し後、制御レジスタへの書き込みは無視されます。

[指定形式]

void	R_UARTMG*n*_Set_PowerOff (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_UARTMG*n*_Send

データの UART 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 UART 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. UART 送信を行う際には、本 API 関数の呼び出し以前に R_UARTMGn_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_UARTMGn_Send ( uint8_t * const tx_buf, uint16_t tx_num );
```

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	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_UARTMG*n*_Receive

データの UART 受信を開始します。

備考 1. 本 API 関数では、 1 バイト単位の UART 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. 実際の UARTF 受信は、本 API 関数の呼び出し後、R_UARTMGn_Start を呼び出すことにより開始されます。

[指定形式]

```
#include        “r_cg_macrodriver.h”
MD_STATUS    R_UARTMGn_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	引数の指定が不正

r_uartmg*n*_callback_sendend

UART 送信転送完了／送信バッファ空き割り込み INTSTMG*n* の発生に伴う処理を行います。

備考　本 API 関数は、UART 送信転送完了／送信バッファ空き割り込み INTSTMG*n* に対応した割り込み処理 r_uartmgn_interrupt_send のコールバック・ルーチン（R_UARTMGn_Send の引数 *tx_num* で指定された数のデータ送信が完了した際の処理）として呼び出されます。

[指定形式]

```
static void    r_uartmgf*n*_callback_sendend ( 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_uartmg*n*_callback_receiveend

UART 受信転送完了割り込み INTSRMG*n* の発生に伴う処理を行います。

備考 本 API 関数は、UART 受信転送完了割り込み INTSRMG*n* に対応した割り込み処理 r_uartmgn_interrupt_receive のコールバック・ルーチン（R_UARTMGn_Receive の引数 *rx_num* で指定された数のデータ受信が完了した際の処理）として呼び出されます。

[指定形式]

```
static void    r_uartmgn_callback_receiveend ( 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_uartmg*n*_callback_error

UART 受信エラーの検出に伴う処理を行います。

備考 本 API 関数は、UART 受信完了割り込み INTSRMG*n* に対応した割り込み処理 r_uartmgn_interrupt_receive または、UART 受信転送完了通信エラー発生割り込み INTSREMG*n* に対応した割り込み処理 r_uartmgn_interrupt_error 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_uartmgn_callback_error ( uint8_t err_type );
```

備考 *n* は、ユニット番号を意味します。

[引数]

I/O	引数	説明
O	uint8_t *err_type*;	UART 受信ステータス割り込みの発生要因 00000xx1B : オーバラン・エラー 00000x1xB : フレーミング・エラー 000001xxB : パリティ・エラー

[戻り値]

なし

r_uartmg*n*_callback_softwareoverrun

オーバラン・エラーの検出に伴う処理を行います。

備考 本 API 関数は、UART 受信転送完了割り込み INTSRMG*n* に対応した割り込み処理 r_uartmgn_interrupt_receive のコールバック・ルーチン（R_UARTMGn_Receive の引数 *rx_num* で指定された数以上のデータを受信した際の処理）として呼び出されます。

[指定形式]

```
#include      “r_cg_macrodriver.h”
static void    r_uartmgn_callback_softwareoverrun ( uint16_t rx_data );
```

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	uint16_t *rx_data*;	受信したデータ（R_UARTMGn_Receive の引数 *rx_num* で指定された数以上に受信したデータ）

[戻り値]

なし

3.3.54 アンプ・ユニット

以下に、コード生成ツールがアンプ・ユニット用として出力する API 関数の一覧を示します。

表 3.56 アンプ・ユニット用 API 関数

API 関数名	機能概要
R_AMP_Create	アンプ・ユニッ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_AMP_Create_UserInit	アンプ・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_AMP_Set_PowerOn	アンプ・ユニット部の電源を投入します。
R_AMP_Set_PowerOff	アンプ・ユニット部の電源を切断します。
R_PGA1_Start	アンプ・ユニット（PGA1）を待機状態にします。
R_PGA1_Stop	アンプ・ユニット（PGA1）を停止します。
R_AMPn_Start	アンプ・ユニット（AMP*n*）を待機状態にします。
R_AMPn_Stop	アンプ・ユニット（AMP*n*）を停止します。

R_AMP_Create

アンプ・ユニットを制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_AMP_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_AMP_Create_UserInit

アンプ・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_AMP_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_AMP_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

R_AMP_Set_PowerOn

アンプ・ユニット部の電源を投入します。

[指定形式]

```
void        R_AMP_Set_PowerOn ( void );
```

[引数]

なし

[戻り値]

なし

R_AMP_Set_PowerOff

アンプ・ユニット部の電源を切断します。

[指定形式]

```
void        R_AMP_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

R_PGA1_Start

アンプ・ユニット（PGA1）を待機状態にします。

[指定形式]

```
void        R_PGA1_Start ( void );
```

[引数]

なし

[戻り値]

なし

R_PGA1_Stop

アンプ・ユニット（PGA1）を停止します。

[指定形式]

```
void        R_PGA1_Stop ( void );
```

[引数]

なし

[戻り値]

なし

R_AMP*n*_Start

アンプ・ユニット（AMP*n*）を待機状態にします。

[指定形式]

void	R_AMP*n*_Start (void);

備考 *n* は、オペアンプユニット番号を意味します。

[引数]

なし

[戻り値]

なし

R_AMP*n*_Stop

アレイ・ユニット（AMP*n*）を停止します。

[指定形式]

void	R_AMP*n*_Stop (void);

備考 *n* は、オペアンプユニット番号を意味します。

[引数]

なし

[戻り値]

なし

3.3.55 データフラッシュライブラリ

以下に、コード生成ツールがデータフラッシュライブラリ用として出力する API 関数の一覧を示します。

表 3.57 データフラッシュライブラリ用 API 関数

API 関数名	機能概要
R_FDL_Create	データフラッシュライブラリを制御するうえで必要となる初期化処理を行います。
R_FDL_Open	データフラッシュライブラリの使用を開始します。
R_FDL_Close	データフラッシュライブラリの使用を終了します。
R_FDL_Write	データをデータフラッシュに書き込みます。
R_FDL_Read	データをデータフラッシュから読み込みます。
R_FDL_Erase	データフラッシュのデータを消去します。

データフラッシュライブラリは、下記の何れかのライブラリがインストールされている必要があります。

- RL78 ファミリ CC-RL コンパイラ用データフラッシュライブラリ Type04 日本リリース版
- RL78 ファミリ CA78K0R コンパイラ用データフラッシュライブラリ Type04 日本リリース版

データフラッシュライブラリのページ：
https://www.renesas.com/products/software-tools/tools/self-programming-library/data-flash-libraries.html
ご使用の前に、データフラッシュライブラリのリリースノートを必ずお読みください。

【注意】コード生成ツールでは、データフラッシュライブラリの下記コマンドを実行する関数をサポートしていません。下記コマンドをご使用になる場合は、「3.2.55.1 データフラッシュライブラリ使用サンプル例（CC-RL）」の(4), (5)を参考にコードを追加してください。

- BLANKCHECK (ブランク・チェック・コマンド)
- IVERIFY (内部ベリファイ・コマンド)

R_FDL_Create

データフラッシュライブラリ Type04 を制御するうえで必要となる初期化処理を行います。

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

[指定形式]

```
void    R_FDL_Create ( void );
```

[引数]

なし

[戻り値]

なし

R_FDL_Open

データフラッシュライブラリを使用するためにドライバをオープンします。
オープン状態で、R_FDL_Write，R_FDL_Read，R_FDL_Erase のコマンドが動作します。

[指定形式]

```
void        R_FDL_Open ( void );
```

[引数]

なし

[戻り値]

なし

R_FDL_Close

データフラッシュライブラリのドライバをクローズします。
再度データフラッシュライブラリを使用するためには、オープン処理（R_FDL_Open）が必要です。

[指定形式]

```
void        R_FDL_Close ( void );
```

[引数]

なし

[戻り値]

なし

R_FDL_Write

データ・フラッシュへの書き込みを行います。
ブランクチェックは行いません。
データ・フラッシュ・メモリの制御状態が、コマンド実行中（PFDL_BUSY）の場合、コマンド実行完了まで書き込みは実行されません。

[指定形式]

```
pfdl_status_t     R_FDL_Write ( pfdl_u16 index, __near pfdl_u08 * buffer, pfdl_u16 bytecount );
```

[引数]

I/O	引数	説明
I	pfdl_u16 *index*;	書き込むデータ・フラッシュのアドレス 0000h ～ 0FFFh
I	pfdl_u08 * *buffer*;	書き込むデータを格納したバッファへのポインタ
I	pfdl_u16 *bytecount*;	書き込むデータのバイト数

[戻り値]

マクロ	説明
PFDL_OK	正常終了
PFDL_BUSY	コマンド実行中
PFDL_ERR_WRITE	書き込みエラー
PFDL_ERR_PARAMETER	引数の指定が不正

R_FDL_Read

データ・フラッシュからの読み込みを行います。

[指定形式]

```
pfdl_status_t     R_FDL_Read ( pfdl_u16 index, __near pfdl_u08 * buffer, pfdl_u16 bytecount );
```

[引数]

I/O	引数	説明
I	pfdl_u16 *index*;	読み込むデータ・フラッシュのアドレス 0000h ～ 0FFFh
I	pfdl_u08 * *buffer*;	読み込むデータを格納するバッファへのポインタ
I	pfdl_u16 *bytecount*;	読み込むデータのバイト数

[戻り値]

マクロ	説明
PFDL_OK	正常終了
PFDL_BUSY	コマンド実行中
PFDL_ERR_PARAMETER	引数の指定が不正

R_FDL_Erase

指定した番号のデータ・フラッシュのブロックを消去します。
戻り値がエラーの場合、対象ブロックに対して書き込みは行えません。
再度、この API を実行し消去が正常に実行されることを確認してください。

[指定形式]

```
pfdl_status_t     R_FDL_Erase ( pfdl_u16 blockno );
```

[引数]

I/O	引数	説明
I	pfdl_u16 *blockno*;	消去するブロックの番号　0 ～ 3

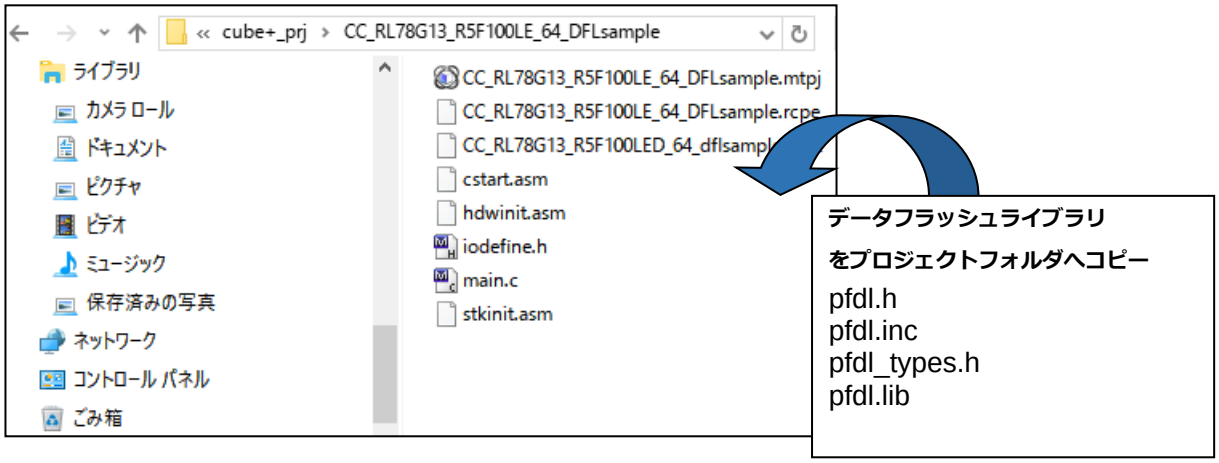
[戻り値]

マクロ	説明
PFDL_OK	正常終了
PFDL_ERR_ERASE	消去エラー
PFDL_ERR_PARAMETER	引数の指定が不正

データフラッシュライブラリ使用サンプル例（CC-RL）

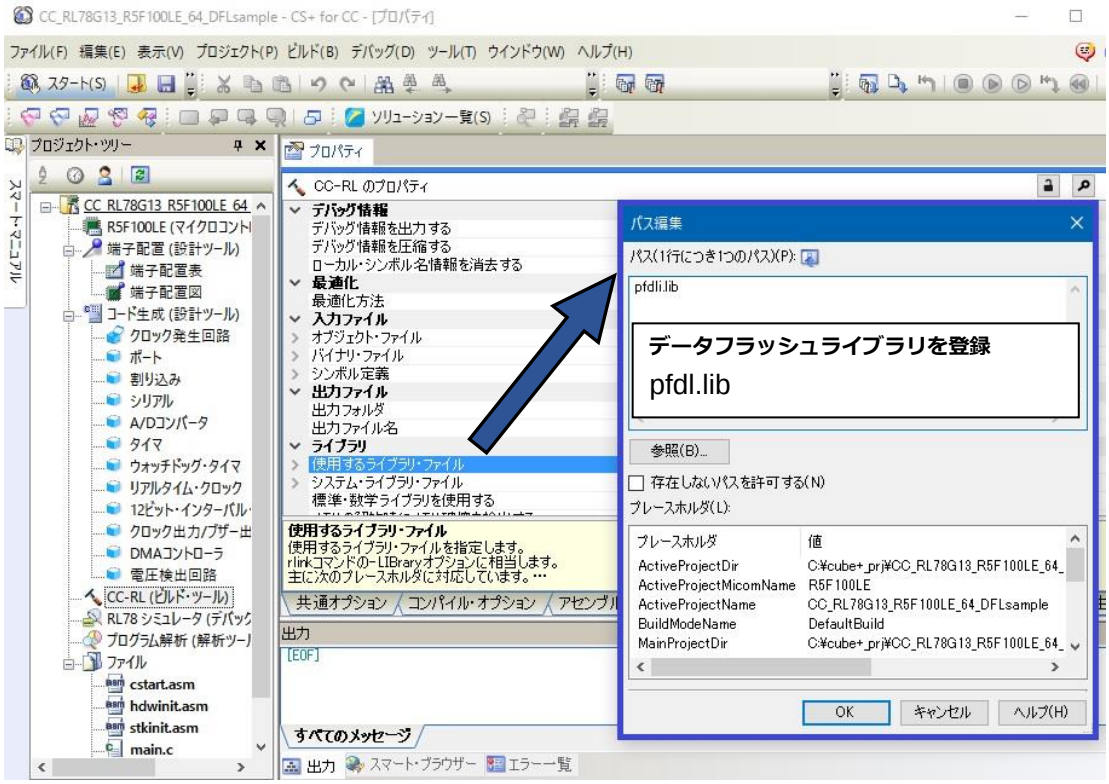
(1) プロジェクトへデータフラッシュライブラリをインストール

図 3.5 データフラッシュライブラリのインストール



(2) データフラッシュライブラリビルドツールへ登録

図 3.6 ビルドツールへ登録



(3) r_main.c を編集（赤文字が追加コード）

図 3.7 r_main.c

```
/***********************************************************************************************************************
Pragma directive
***********************************************************************************************************************/
/* Start user code for pragma. Do not edit comment generated here */
pfdl_status_t result;
uint8_t loop;
static pfdl_u08 gtBuffer[] = { 0x11, 0x12, 0x13, 0x14, 0x55, 0xAA, 0xFF, 0x00 };
static pfdl_u08 gtReadBuffer[ 128 ];
/* End user code. Do not edit comment generated here */
        省略
void main(void)
        省略
/***********************************************************************************************************************
* Function Name: R_MAIN_UserInit
* Description  : This function adds user code before implementing main function.
* Arguments    : None
* Return Value : None
***********************************************************************************************************************/
void R_MAIN_UserInit(void)
{
    /* Start user code. Do not edit comment generated here */
    EI();
    R_FDL_Create();
    R_FDL_Open();
    /* データフラッシュのブランクチェック */
    for ( loop=0; loop<10; loop++)
    {
        result = R_FDL_BLANKCHECK( loop * 8, gtBuffer, 8 );
        if ( result == PFDL_ERR_MARGIN )
        {
            result = R_FDL_Erase( 0 );
        }
    }
    /* データフラッシュへの書き込み */
    for ( loop=0; loop<10; loop++)
    {
        gtBuffer[ 7 ] = loop;
        result = R_FDL_Write( loop * 8, gtBuffer, 8 );
        if ( result != PFDL_OK )
        {
            break;
        }
    }
    /* データフラッシュの内部ベリファイ */
    for ( loop=0; loop<10; loop++)
    {
        result = R_FDL_IVERIFY( loop * 8, gtBuffer, 8 );
        if (result == PFDL_ERR_MARGIN )
        {
            break;
        } else {
            /* データフラッシュからの読み込み */
            result = R_FDL_Read( loop * 8, &gtReadBuffer[ loop * 8], 8 );
            if ( result != PFDL_OK )
            {
                break;
            }
        }
    }
    /* データフラッシュの消去 */
```

```
    result = R_FDL_Erase( 0 );
    R_FDL_Close();
    /* End user code. Do not edit comment generated here */
}
```

(4) r_cg_pfdl.h を編集（赤文字が追加コード）

図 3.8 r_cg_pfdl.h

```
/***********************************************************************************************************************
Global functions
***********************************************************************************************************************/
void R_FDL_Create(void);
pfdl_status_t R_FDL_Write(pfdl_u16 index, __near pfdl_u08* buffer, pfdl_u16 bytecount);
pfdl_status_t R_FDL_Read(pfdl_u16 index, __near pfdl_u08* buffer, pfdl_u16 bytecount);
pfdl_status_t R_FDL_Erase(pfdl_u16 blockno);
pfdl_status_t R_FDL_BLANKCHECK (pfdl_u16 index, pfdl_u16 bytecount);
pfdl_status_t R_FDL_IVERIFY (pfdl_u16 index, pfdl_u16 bytecount);
void R_FDL_Open(void);
void R_FDL_Close(void);
```

(5) r_cg_pfdl.c を編集（赤文字が追加コード）

図 3.9 r_cg_pfdl.c

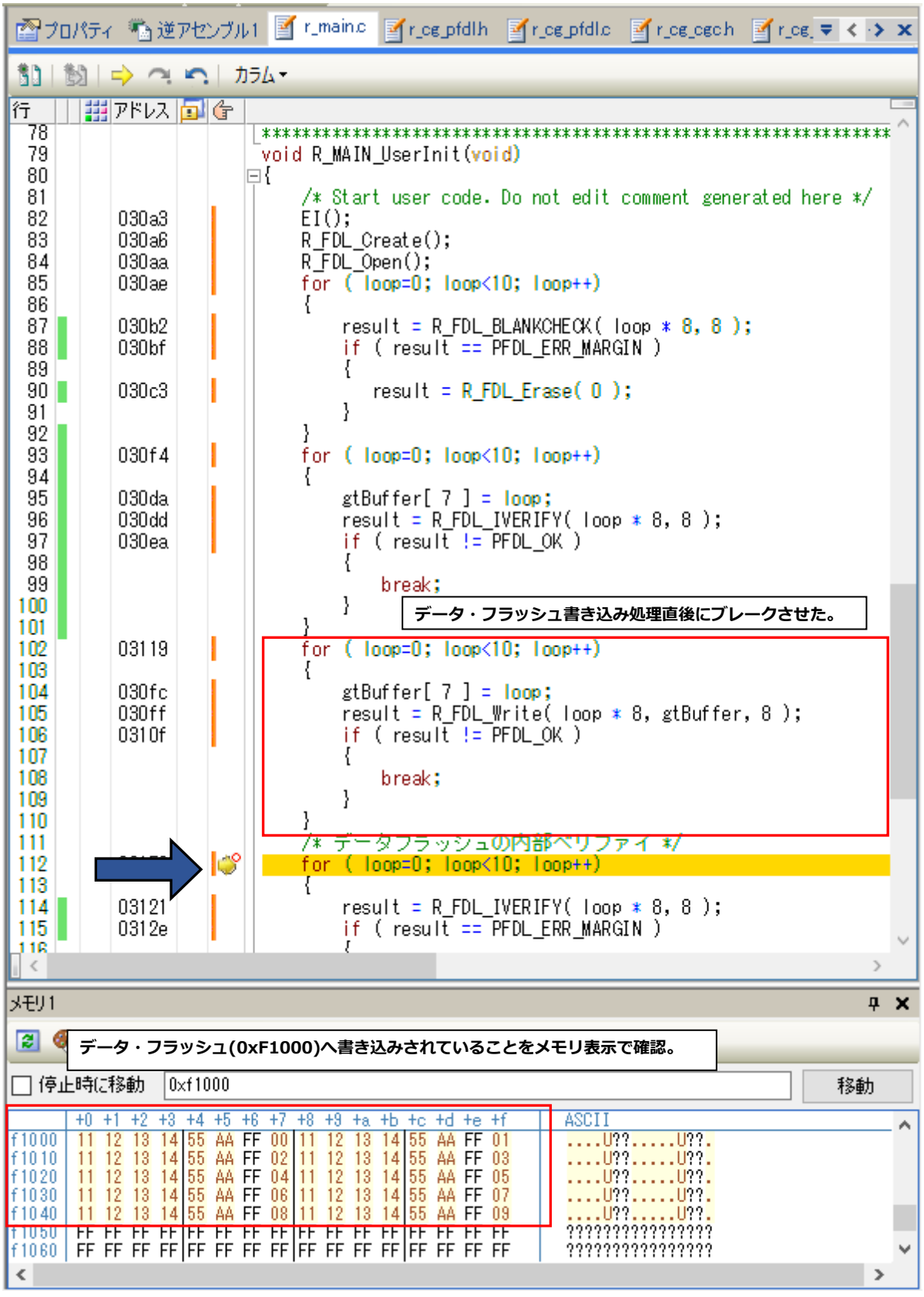
```
/***********************************************************************************************************************
* Function Name: R_FDL_Close
* Description  : This function closes the RL78 data flash library.
* Arguments    : None
* Return Value : None
***********************************************************************************************************************/
void R_FDL_Close(void)
{
    PFDL_Close();
    gFdlStatus = 0;
}

/* Start user code for adding. Do not edit comment generated here */
***********************************************************************************************************************
* Function Name: R_FDL_BLANKCHECK
* Description  : This function blank check a data to the RL78 data flash memory.
* Arguments    : index -
*                    It is destination address of Flash memory for blank check. The address range is from
*                    0x0000 to 0x0FFF
*                buffer -
*                    The top address of data to blank check
*                bytecount -
*                    The size of data to blank check (Unit is byte)
* Return Value : pfdl_status_t -
*                    status of blank check command
***********************************************************************************************************************/
pfdl_status_t R_FDL_BLANKCHECK(pfdl_u16 index, pfdl_u16 bytecount)
{
    if (gFdlStatus == 1)
    {
        gFdlReq.index_u16     = index;
        gFdlReq.bytecount_u16 = bytecount;
        gFdlReq.command_enu   = PFDL_CMD_BLANKCHECK_BYTES;
        gFdlResult = PFDL_Execute(&gFdlReq);
        /* Wait for completing command  */
        while(gFdlResult == PFDL_BUSY)
        {
            NOP();
            NOP();
```

```
            gFdlResult = PFDL_Handler();        /* The process for confirming end */
        }
    }
    else
    {
        gFdlResult = PFDL_ERR_PROTECTION;
    }
    return gFdlResult;
}
***********************************************************************************************************************
* Function Name: R_FDL_IVERIFY
* Description  : This function performs internal verification on the execution range area.
* Arguments    : index -
*                    It is destination address of Flash memory for iverify a data. The address range is from
*                    0x0000 to 0x0FFF
*                buffer -
*                    The top address of data to iverify
*                bytecount -
*                    The size of data to iverify (Unit is byte)
* Return Value : pfdl_status_t -
*                    status of iverify command
***********************************************************************************************************************/
pfdl_status_t R_FDL_IVERIFY(pfdl_u16 index, pfdl_u16 bytecount)
{
    if (gFdlStatus == 1)
    {
        gFdlReq.index_u16     = index;
        gFdlReq.bytecount_u16 = bytecount;
        gFdlReq.command_enu   = PFDL_CMD_IVERIFY_BYTES;
        gFdlResult = PFDL_Execute(&gFdlReq);
        /* Wait for completing command  */
        while(gFdlResult == PFDL_BUSY)
        {
            NOP();
            NOP();
            gFdlResult = PFDL_Handler();        /* The process for confirming end */
        }
    }
    else
    {
        gFdlResult = PFDL_ERR_PROTECTION;
    }
    return gFdlResult;
}
/* End user code. Do not edit comment generated here */
```

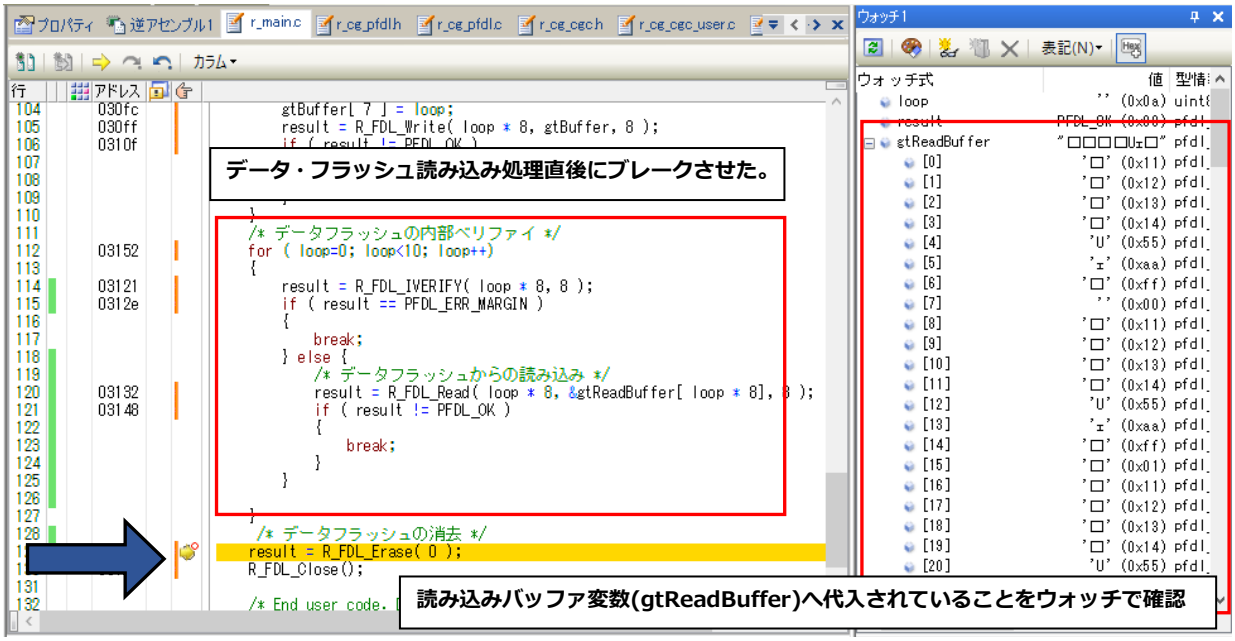
(6) QB-R5F100LE-TB で書き込み動作確認

図 3.10 データフラッシュ書き込み確認



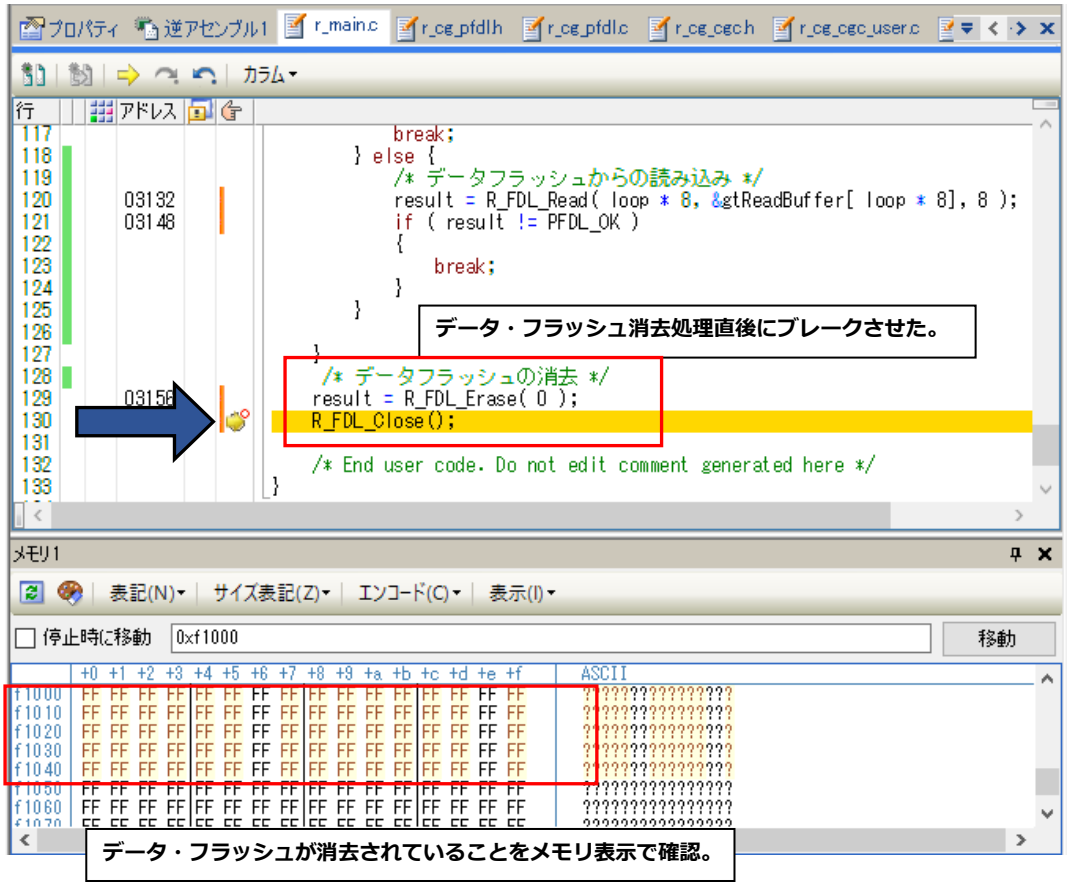
(7) QB-R5F100LE-TB で読み込み動作確認

図 3.11 データフラッシュ読み込み確認



(8) QB-R5F100LE-TB で動作確認(消去)

図 3.12 データフラッシュ消去確認



改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2018.07.01	ー	初版発行
1.01	2019.01.01	410	備考 2 の誤記を修正
1.02	2019.10.01	6	「1.3 対応コンパイラ」を追加
		6	「1.4 注意」を追加
		9	表 2.1 の共通にファイルと API 関数を追加
		24	「3.2 初期化処理」を追加
		28, 29	3.3.1 に__low_level_init, HardwareSetuo を追加
		63	R_TAU*m*_Channel*n*_Start の機能概要に備考 2 を追加
		66	R_TAU*m*_Channel*n*_Stop の機能概要に備考を追加
		88	R_TMR_RJ0_Get_PulseWidth の機能概要に備考 3 を追加
		95	R_TMRJ0_Get_PulseWidth の機能概要に備考 3 を追加
		210	R_RTC_Set_CounterValue の機能概要に備考を追加
		252	R_IT_Start の機能概要に備考を追加
		253	R_IT_Stop の機能概要に備考を追加
		416	r_csi*mn*_callback_sendend の機能概要に備考 2 を追加
		417	r_csi*mn*_callback_receiveend の機能概要に備考 2 を追加
		473	R_IICA*n*_Master_Send の引数備考に adr 説明(図)を追加
		474	R_IICA*n*_Master_Receive の引数備考に adr 説明(図)を追加
		732	データフラッシュライブラリ使用前の注意事項を追加
		739	データフラッシュライブラリ使用サンプル例を追加
1.03	2020.12.20	11, 257-264	8 ビット・インターバル・タイマの API 関数名の誤記を修正
		8,100, 120,121	R_TMRD_PWMOP_ForcedOutput_Stop, R_TMRD_PWMOP_Set_PowerOff を追加
		420	r_iic*mn*_interrupt の機能概要に備考 2 を追加
		424	R_IIC*mn*_Master_Send の機能概要に備考 2 を追加
		425	R_IIC*mn*_Master_Receive の機能概要に備考 2 を追加
		428	r_iic*mn*_callback_master_error の引数の説明を追加
		473	R_IICA*n*_Master_Send の戻り値の誤記修正
		474	R_IICA*n*_Master_Receive の戻り値の誤記修正
		475	r_iica*n*_callback_master_sendend の機能概要に備考 2 を追加
		476	r_iica*n*_callback_master_receiveend の機能概要に備考 2 を追加
		479	R_IICA*n*_Slave_Receive の戻り値の誤記修正
1.04	2021.10.15	422	R_IIC*mn*_StopCondition の機能概要に備考を追加
1.05	2024.7.22	5, 6	「1.1 概要」、「1.2 特長」、「1.3 対応コンパイラ」を更新
		7	「表 2.1 出力ファイル(1/15)」の「共通」を更新
		24	「3.2.2 GNU/LLVM コンパイラ用」を更新

コード生成ツール ユーザーズマニュアル
RL78 APIリファレンス編

発行年月日	2018年7月1日	Rev.1.00
	2019年1月1日	Rev.1.01
	2019年10月1日	Rev1.02
	2020年12月20日	Rev1.03
	2021年10月15日	Rev1.04
	2024年7月22日	Rev1.05

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）

コード生成ツール

ルネサス エレクトロニクス株式会社

R20UT4323JJ0105