User's Manual

CubeSuite+ V2.01.00

統合開発環境

ユーザーズマニュアル　RL78 設計編

対象デバイス

RL78 ファミリ

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

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　2013.09

ご注意書き

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

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

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

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

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

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

(2012.4)

このマニュアルの使い方

このマニュアルは，RL78 ファミリ用アプリケーション・システムを開発する際の統合開発環境である CubeSuite+ について説明します。

CubeSuite+ は，RL78 ファミリの統合開発環境（ソフトウエア開発における，設計，実装，デバッグなどの各開発フェーズに必要なツールをプラットフォームである IDE に統合）です。統合することで，さまざまなツールを使い分ける必要がなく，本製品のみを使用して開発のすべてを行うことができます。

対 象 者　このマニュアルは，CubeSuite+ を使用してアプリケーション・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，CubeSuite+ の持つソフトウエア機能をユーザに理解していただき，これらのデバイスを使用するシステムのハードウエア，ソフトウエア開発の参照用資料として役立つことを目的としています。

構　　成　このマニュアルは，大きく分けて次の内容で構成しています。

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

凡　　例

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

関連資料　関連資料は暫定版の場合がありますが，この資料では「暫定」の表示をしておりません。あらかじめご了承ください。

資料名		資料番号	
		和文	英文
CubeSuite+ 統合開発環境 ユーザーズ・マニュアル	起動編	R20UT2682J	R20UT2682E
	RX 設計編	R20UT2683J	R20UT2683E
	V850 設計編	R20UT2134J	R20UT2134E
	RL78 設計編	このマニュアル	R20UT2684E
	78K0R 設計編	R20UT2137J	R20UT2137E
	78K0 設計編	R20UT2138J	R20UT2138E
	RH850 コーディング編	R20UT2584J	R20UT2584E
	RX コーディング編	R20UT2470J	R20UT2470E
	V850 コーディング編	R20UT0553J	R20UT0553E
	コーディング編（CX コンパイラ）	R20UT2659J	R20UT2659E
	RL78, 78K0R コーディング編	R20UT2140J	R20UT2140E
	78K0 コーディング編	R20UT2141J	R20UT2141E
	RH850 ビルド編	R20UT2585J	R20UT2585E
	RX ビルド編	R20UT2472J	R20UT2472E
	V850 ビルド編	R20UT0557J	R20UT0557E
	ビルド編（CX コンパイラ）	R20UT2142J	R20UT2142E
	RL78, 78K0R ビルド編	R20UT2143J	R20UT2143E
	78K0 ビルド編	R20UT0783J	R20UT0783E
	RH850 デバッグ編	R20UT2685J	R20UT2685E
	RX デバッグ編	R20UT2702J	R20UT2702E
	V850 デバッグ編	R20UT2446J	R20UT2446E
	RL78 デバッグ編	R20UT2445J	R20UT2445E
	78K0R デバッグ編	R20UT0732J	R20UT0732E
	78K0 デバッグ編	R20UT0731J	R20UT0731E
	解析編	R20UT2686J	R20UT2686E
	メッセージ編	R20UT2687J	R20UT2687E

注意　上記関連資料は，予告なしに内容を変更することがあります。設計などには，必ず最新の資料を使用してください。

目　　次

第1章　概　　説

CubeSuite+ は，アプリケーション・システムを開発する際の統合開発環境であり，設計／コーディング／ビルド／デバッグなどといった一連の作業を実施することができます。

本章では，設計ツール（端子配置／コード生成）の概要について説明します。

1.1　概　　要

設計ツールは，CubeSuite+ が提供しているコンポーネントの 1 種であり，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。

1.2　特　　長

以下に，設計ツール（端子配置／コード生成）の特長を示します。

- コード生成機能

  コード生成では，GUI ベースで設定した情報に応じたデバイス・ドライバ・プログラムを出力するだけでなく，main 関数を含んだサンプル・プログラム，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／コード生成を用いて設定した情報を各種形式のファイルで出力し，設計資料として利用することができます。

- リネーム機能

  コード生成が出力するファイル名，およびソース・コードに含まれている API 関数の関数名については，デフォルトの名前が付与されますが，ユーザ独自の名前に変更することもできます。

第２章　機能（端子配置）

本章では，設計ツール（端子配置）が提供している主な機能を操作手順とともに説明します。

備考　本章では，対象デバイスが“RL78/G13（ROM：16KB）R5F1006A（20pin）”の場合を例にとり，主な機能の説明を行っています。

2.1　概　　要

端子配置は，マイクロコントローラの端子配置状況を入力することにより，端子配置表，端子配置図といったレポート・ファイルを出力させることができます。

なお，端子配置の操作手順は，以下のとおりです。

（1）CubeSuite+ の起動

Windows の［スタート］メニューから CubeSuite+ を起動します。

備考　“CubeSuite+ の起動”についての詳細は，「CubeSuite+ 統合開発環境 ユーザーズマニュアル 起動編」を参照してください。

（2）プロジェクトの作成／読み込み

プロジェクトの新規作成（プロジェクトの種類，使用するマイクロコントローラ，使用するビルド・ツールなどの定義），または既存のプロジェクトの読み込みを行います。

備考　“プロジェクトの作成／読み込み”についての詳細は，「CubeSuite+ 統合開発環境 ユーザーズマニュアル 起動編」を参照してください。

（3）端子配置表 パネルのオープン

マイクロコントローラの各端子に関する情報を記述するための端子配置表 パネルをオープンします。

（a）表示項目の選択

端子配置表に表示する項目を選択します。

（b）表示順序の変更

端子配置表に表示する項目の順序を変更します。

（c）列の追加

端子配置表に対する列の追加を行います。

（d）列の削除

端子配置表に対する列の削除を行います。

（4）端子配置図 パネルのオープン

端子に関する情報の記述状況を確認するための端子配置図 パネルをオープンします。

（a）マイクロコントローラの形状選択

端子配置図 パネルに表示するマイクロコントローラの形状を選択します。

（b）表示色の選択

端子配置図 パネルの各端子（電源端子，特殊端子，使用端子など）に関する情報の記述状況を確認するための表示色を選択します。

（c）ポップアップ情報の選択

端子配置図 パネルの各端子上にマウス・カーソルを移動した際，ポップアップ表示させる情報の種類を選択します。

（d）付加情報の選択

端子配置図 パネルの端子部分に表示させる情報の種類を選択します。

（5）情報の記述

端子配置表 パネルでマイクロコントローラの各端子に関する情報を記述します。

（6）レポート・ファイルの出力

レポート・ファイル（端子配置を用いて設定した情報を保持したファイル：端子配置表，端子配置図）を指定されたフォルダに出力します。

（a）端子配置表の出力

端子配置表を出力します。

（b）端子配置図の出力

端子配置図を出力します。

（7）プロジェクトの保存

プロジェクトの保存を行います。

備考 “プロジェクトの保存”についての詳細は，「CubeSuite+ 統合開発環境 ユーザーズマニュアル 起動編」を参照してください。

2. 2　端子配置表 パネルのオープン

マイクロコントローラの各端子に関する情報を記述するための端子配置表 パネルをオープンします。

なお，端子配置表 パネルのオープンは，プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［端子配置（設計ツール）］→［端子配置表］をダブルクリックすることにより行います。

図 2—1　端子配置表 パネルのオープン

備考 1. 端子配置が未対応のマイクロコントローラがプロジェクトで定義された場合，プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］に“［端子配置（設計ツール）］ノード”は表示されません。

2. 端子配置表 パネルは 3 個のタブから構成され，タブを選択することにより，“マイクロコントローラの各端子に関する情報”の表示順序が切り替わります。

- ［端子番号］タブ

  マイクロコントローラの各端子に関する情報を端子番号順で表示

- ［マクロ］タブ

  マイクロコントローラの各端子に関する情報を周辺機能単位にグルーピングされた順序で表示

- ［外部周辺］タブ

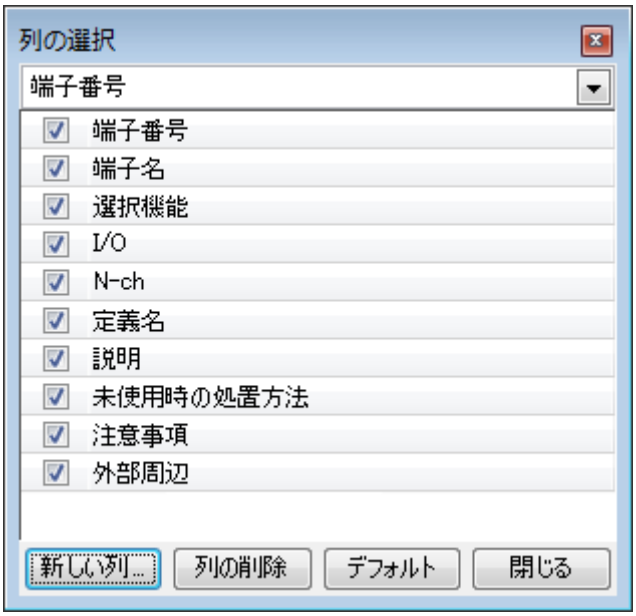
  外部周辺に接続された端子に関する情報を外部周辺部品単位にグルーピングされた順序で表示

2. 2. 1 表示項目の選択

端子配置では，端子配置表の左上に設けられた  ボタンで端子配置表の表示項目を選択することができます。

なお，表示項目の選択は，端子配置表の左上に設けられた  ボタンをクリックすることによりオープンする列の選択 ダイアログで行います。

図 2—2 表示項目の選択



備考 表示項目の選択は，該当チェック・ボックスをクリックすることにより行います。

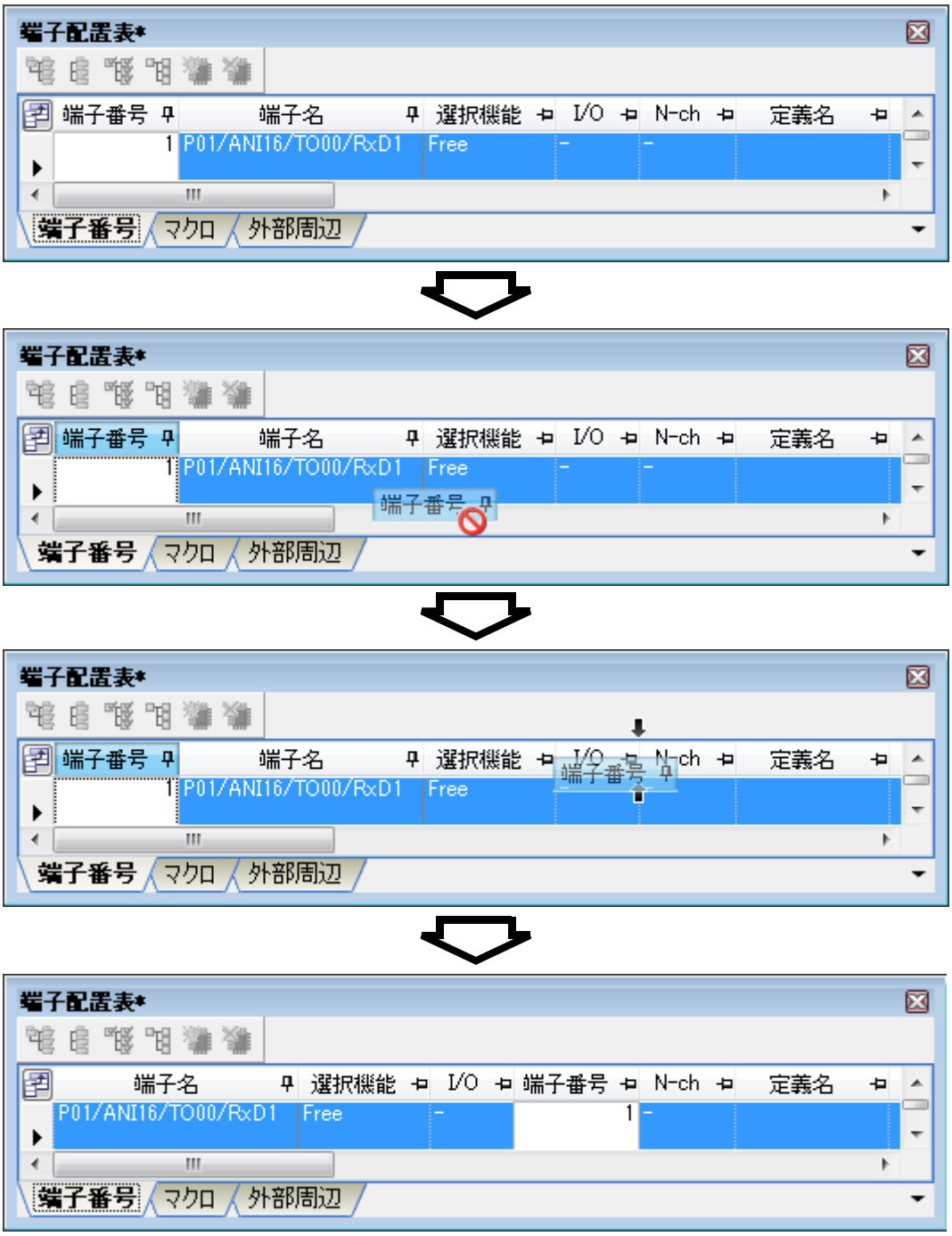
表 2—1 表示項目の選択

チェック状態	該当項目を端子配置表に表示します。
非チェック状態	該当項目を端子配置表から非表示とします。

2. 2. 2　表示順序の変更

端子配置では，端子配置表の列をドラッグしたのち，移動先にドロップすることにより，表示項目の表示順序を変更（列を移動）することができます。

図 2―3　表示順序の変更



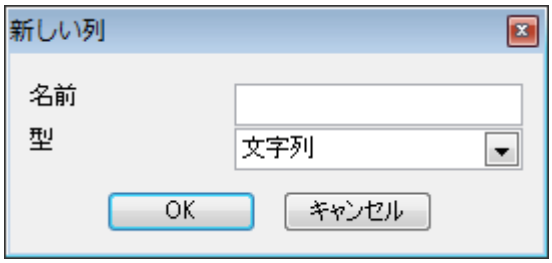
備考　表示順序の変更は，端子配置表の左上に設けられた ボタンをクリックすることによりオープンする列の選択 ダイアログの表示項目選択エリアに表示されている項目をドラッグしたのち，端子配置表の移動先にドロップすることでも，表示項目の表示順序を変更することができます。

2. 2. 3 列の追加

端子配置では，端子配置表の左上に設けられたボタンをクリックすることによりオープンする列の選択 ダイアログの［新しい列 ...］ボタンで“ユーザ独自の列”を端子配置表に追加することができます。

なお，列の追加は，列の選択 ダイアログの［新しい列 ...］ボタンをクリックすることによりオープンする新しい列 ダイアログで行います。

図 2—4 列の追加



備考 端子配置表“［マクロ］タブ，［外部周辺］タブの第 1 階層”については，列の追加が制限されています。

2. 2. 4 列の削除

端子配置では，端子配置表の左上に設けられたボタンをクリックすることによりオープンする列の選択 ダイアログの［列の削除］ボタンで“ユーザ独自の列”を端子配置表から削除することができます。

なお，列の削除は，列の選択 ダイアログの表示項目選択エリアで削除対象列を選択したのち，［列の削除］ボタンをクリックすることにより行います。

図 2—5 列の削除

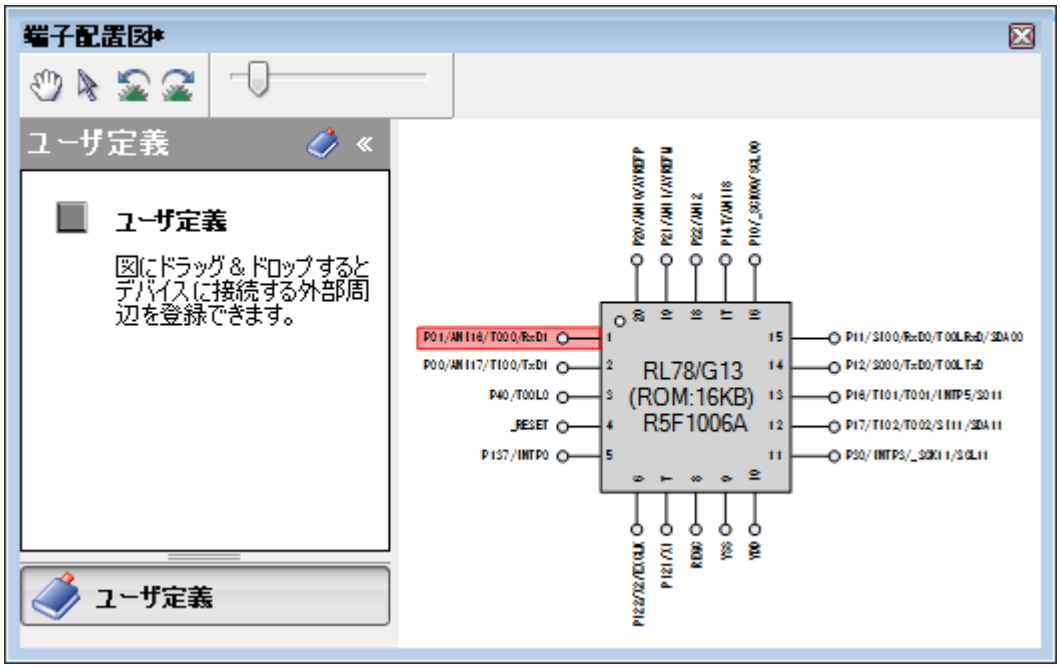
備考 削除可能な列は，新しい列 ダイアログでユーザが独自に追加した列に限られます。

2. 3　端子配置図 パネルのオープン

マイクロコントローラの各端子に関する情報の記述状況を確認するための端子配置図 パネルをオープンします。

なお，端子配置図 パネルのオープンは，プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［端子配置（設計ツール）］→［端子配置図］をダブルクリックすることにより行います。

図 2―6　端子配置図 パネルのオープン



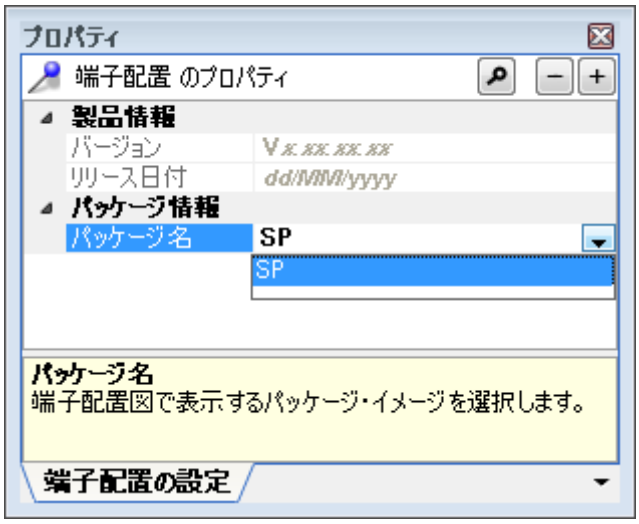
備考　プロパティ パネルの［端子配置の設定］タブでパッケージ名に“BGA”を選択している場合，端子配置図 パネルをオープンすることができません。

2. 3. 1　マイクロコントローラの形状選択

「2. 3　端子配置図 パネルのオープン」でオープンした端子配置図 パネルに表示するマイクロコントローラの形状を選択します。

なお，マイクロコントローラの形状選択は，プロパティ パネルの［端子配置の設定］タブ→［パッケージ情報］→［パッケージ名］で該当形状を選択することにより行います。

図 2―7　マイクロコントローラの形状選択



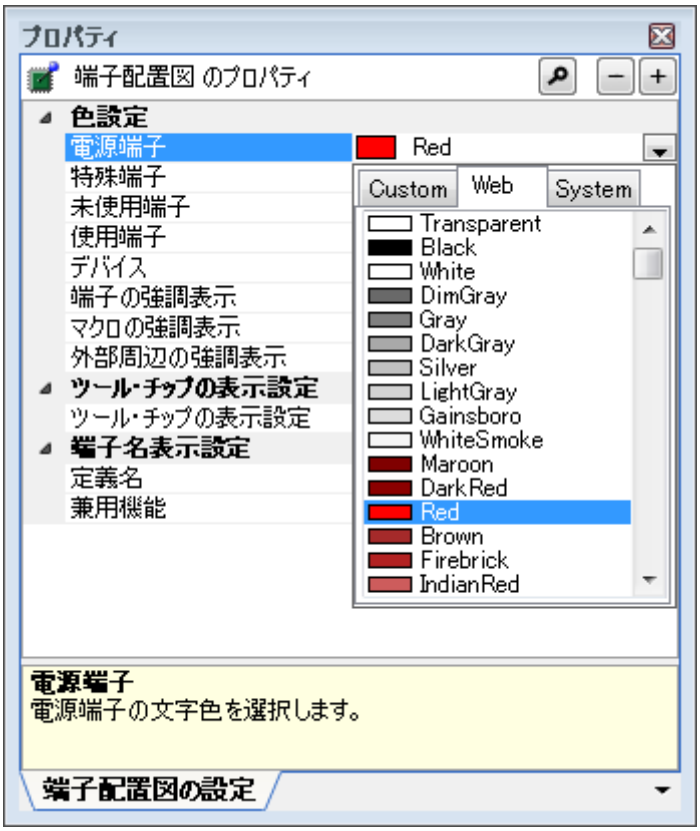
備考　マイクロコントローラの形状選択は，オーダ名称（GC，GF など）で行います。

2. 3. 2 表示色の選択

「2. 3 端子配置図 パネルのオープン」でオープンした端子配置図 パネルの各端子（電源端子，特殊端子，未使用端子など）に関する情報の記述状況を確認するための表示色を選択します。

なお，表示色の選択は，プロパティ パネルの［端子配置図の設定］タブ→［色設定］で該当色を選択することにより行います。

図 2—8 表示色の選択



備考 表示色の選択は，以下の 8 種類に対して行います。

表 2—2 表示色の選択

設定対象	概要
電源端子	電源端子（用途が電源に限定されている端子）の表示色を選択します。
特殊端子	特殊端子（用途が規定されている端子）の表示色を選択します。
未使用端子	未使用端子（端子配置表 パネルにおいて，用途が未設定の兼用端子）の表示色を選択します。
使用端子	使用端子（端子配置表 パネルにおいて，用途が設定済みの兼用端子）の表示色を選択します。
デバイス	マイクロコントローラ本体部の表示色を選択します。

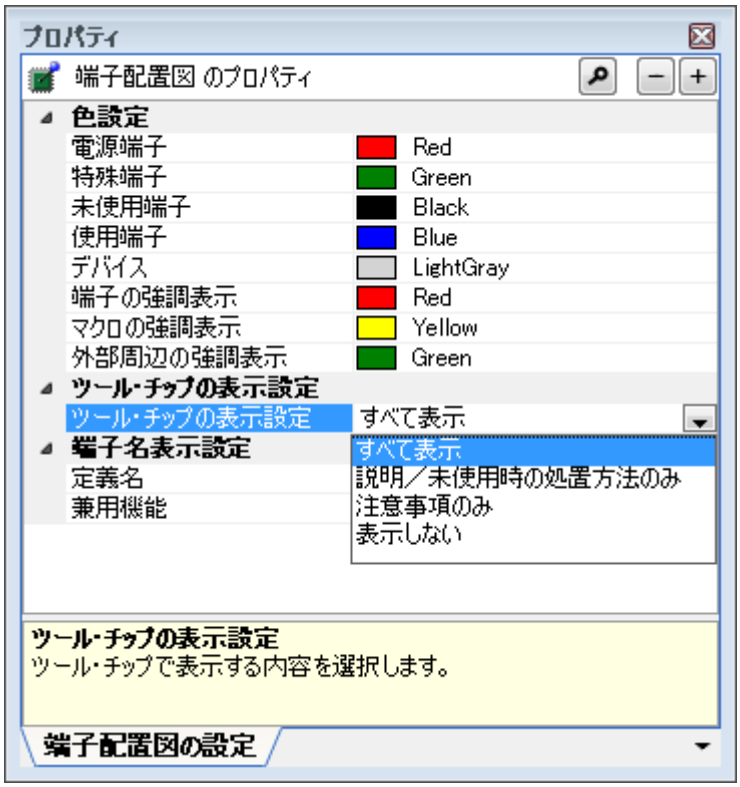
設定対象	概要
端子の強調表示	端子配置表 パネルの［端子番号］タブで選択された項目に対応した端子の背景色を選択します。
マクロの強調表示	端子配置表 パネルの［マクロ］タブで選択された項目に対応した端子の背景色を選択します。
外部周辺の強調表示	端子配置表 パネルの［外部周辺］タブで選択された項目に対応した端子の背景色を選択します。

2. 3. 3 ポップアップ情報の選択

「2. 3 端子配置図 パネルのオープン」でオープンした端子配置図 パネルの各端子上にマウス・カーソルを移動した際にポップアップ表示させる情報の種類を選択します。

なお，ポップアップ情報の選択は，プロパティ パネルの［端子配置図の設定］タブ→［ツール・チップの表示設定］→［ツール・チップの表示設定］で該当種類を選択することにより行います。

図 2—9 ポップアップ情報の選択



備考 ポップアップ情報の選択は，以下の 4 種類から行います。

表 2—3 ポップアップ情報の選択

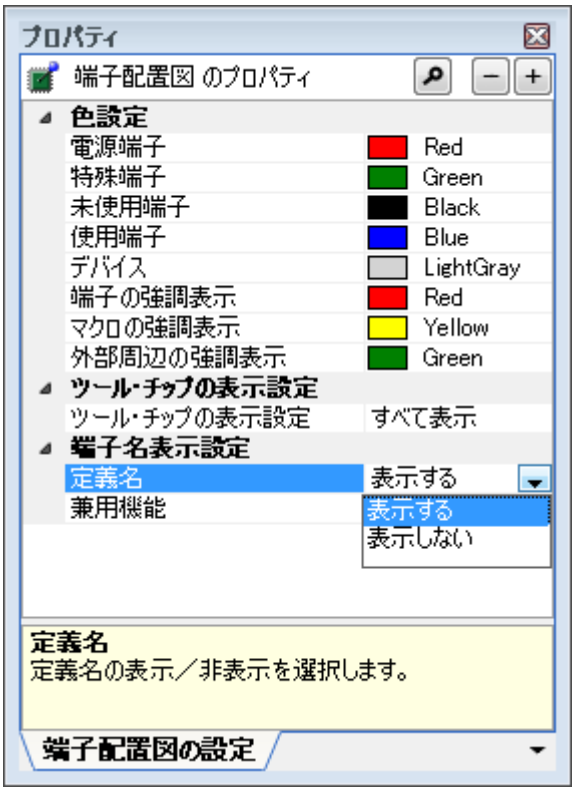
ポップアップ情報	概要
すべて表示	端子配置表の“説明”，“未使用時の処置方法”，“注意事項”に記載されている文字列を表示します。
説明／未使用時の処置方法のみ	端子配置表の“説明”，“未使用時の処置方法”に記載されている文字列を表示します。
注意事項のみ	端子配置表の“注意事項”に記載されている文字列を表示します。
表示しない	端子上にマウス・カーソルを移動しても，何も表示しません。

2. 3. 4 付加情報の選択

「2. 3 端子配置図 パネルのオープン」でオープンした端子配置図 パネルの端子部分に表示させる情報の種類を選択します。

なお，付加情報の選択は，プロパティ パネルの［端子配置図の設定］タブ→［端子名表示設定］で該当情報を選択することにより行います。

図 2—10 付加情報の選択



備考 1. 定義名（端子配置表の“定義名”に記載された文字列を付与した形式で表示するか否か）については，以下の 2 種類から選択します。

表示する	端子配置表の“定義名”に記載されている文字列を付与した形式で表示します。
表示しない	端子配置表の“定義名”に記載されている文字列を付与しません。

2. 兼用機能（端子配置表の“選択機能”で機能を選択した際，非選択機能についても表示するか否か）については，以下の 2 種類から選択します。

すべて	端子配置表の“選択機能”で選択された機能をかっこで括った形式で表示します。
選択機能のみ	端子配置表の“選択機能”で選択された機能のみを端子配置図に表示します。

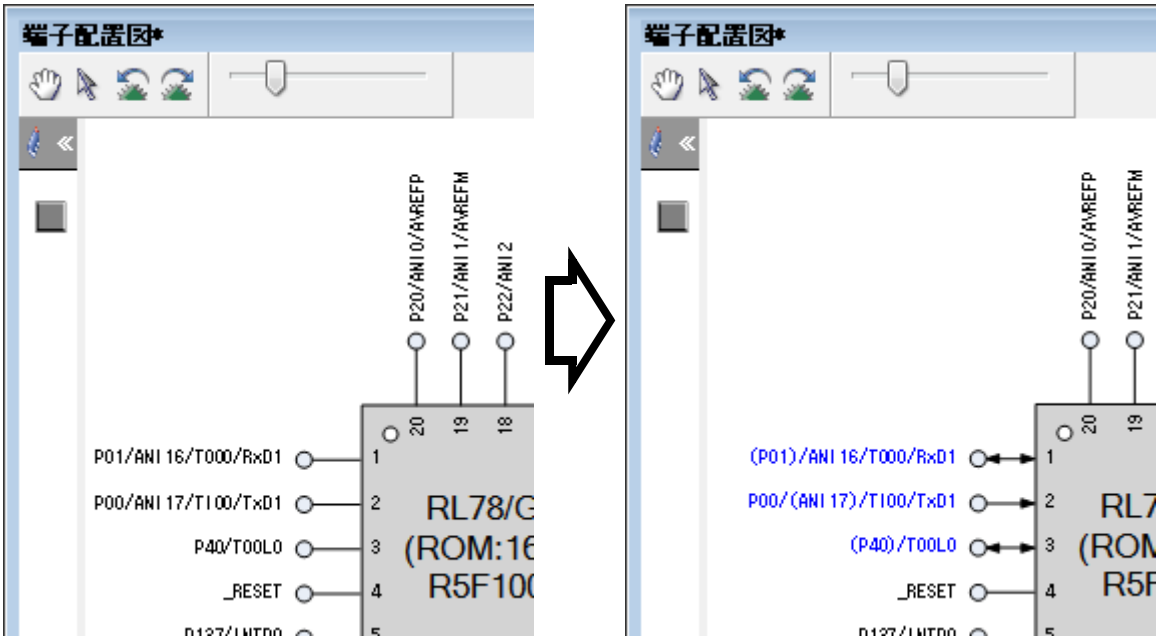
2.4　情報の記述

「2.2　端子配置表 パネルのオープン」でオープンした端子配置表 パネルでマイクロコントローラの各端子に関する情報を記述します。

備考 1. 端子配置表の“端子番号”，“端子名”，“説明”，“未使用時の処置方法”，“注意事項”については，固定化された情報のため，該当欄に情報を追記することはできません。

2. “選択機能”欄の Free を固有端子名に変更した場合，端子配置図 パネルの該当端子色がプロパティ パネルの［端子配置図の設定］タブ→［色設定］で選択された“未使用端子の表示色”から“使用端子の表示”へと変化します。

図 2—11　表示色の変化



2. 5　レポート・ファイルの出力

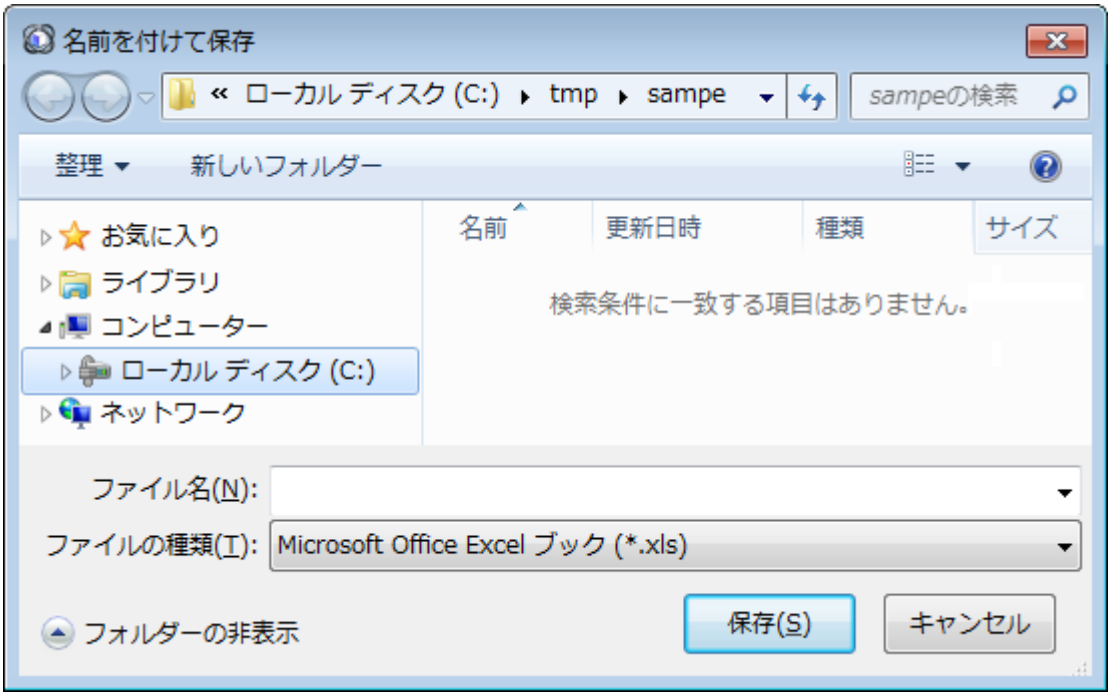
レポート・ファイル（端子配置を用いて設定した情報を保持したファイル：端子配置表，端子配置図）を指定されたフォルダに出力します。

2. 5. 1　端子配置表の出力

［ファイル］メニュー→［名前を付けて 端子配置表 を保存 ...］を選択し，レポート・ファイル（端子配置を用いて設定した情報を保持したファイル：端子配置表）を出力します。

なお，端子配置表の出力先は，［ファイル］メニュー→［名前を付けて 端子配置表 を保存 ...］を選択することによりオープンする名前を付けて保存 ダイアログで指定されたフォルダとなります。

図 2―12　端子配置表の出力



備考 1.　すでに端子配置表が出力されていた場合，［ファイル］メニュー→［端子配置表 を保存］を選択することにより，該当表を上書きします。

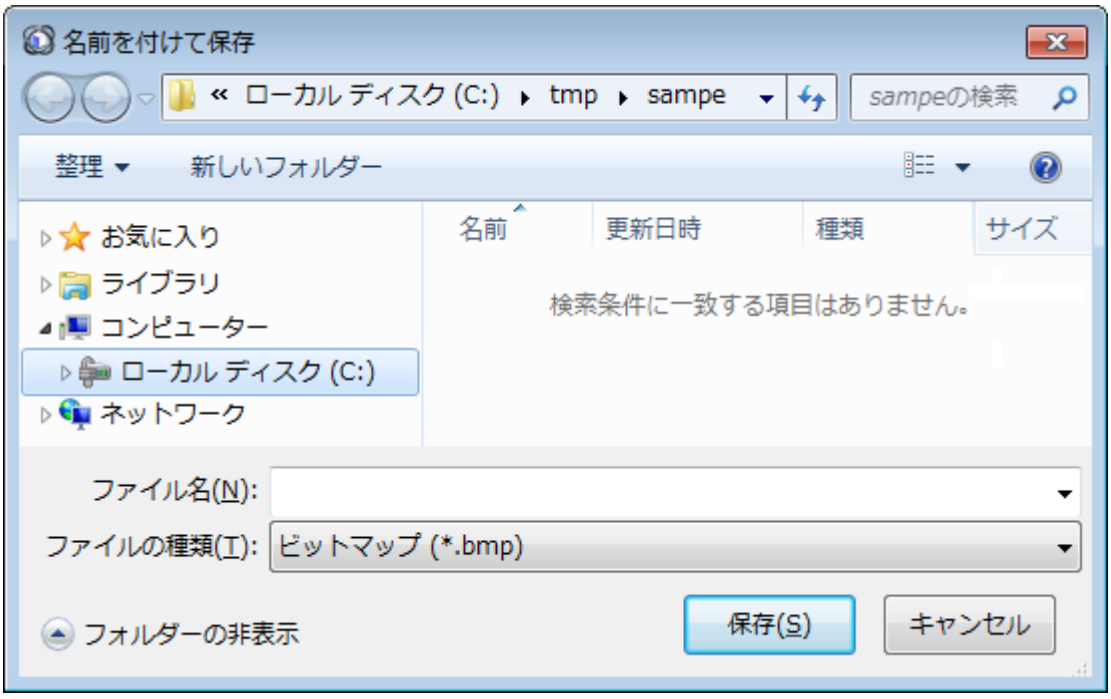
2.　端子配置表の出力形式は，Microsoft Office Excel ブック形式に限られます。

2. 5. 2 端子配置図の出力

［ファイル］メニュー→［名前を付けて 端子配置図 を保存 ...］を選択し，レポート・ファイル（端子配置を用いて設定した情報を保持したファイル：端子配置図）を出力します。

なお，端子配置図の出力先は，［ファイル］メニュー→［名前を付けて 端子配置図 を保存 ...］を選択することによりオープンする名前を付けて保存 ダイアログで指定されたフォルダとなります。

図 2—13 端子配置図の出力



備考 すでに端子配置図が出力されていた場合，［ファイル］メニュー→［端子配置図 を保存］を選択することにより，該当図を上書きします。

第３章　機能（コード生成）

本章では，設計ツール（コード生成）が提供している主な機能を操作手順とともに説明します。

備考　本章では，対象デバイスが“RL78/L13（ROM：128KB）R5F10WMG（80pin）”の場合を例にとり，主な機能の説明を行っています。

3.1　概　　要

コード生成は，デバイス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）を制御する際に必要な情報を CubeSuite+ のパネル上で選択／入力することにより，対応するソース・コード（デバイス・ドライバ・プログラム）を出力します。

なお，コード生成の操作手順は，以下のとおりです。

（1）CubeSuite+ の起動

Windows の［スタート］メニューから CubeSuite+ を起動します。

備考　“CubeSuite+ の起動”についての詳細は，「CubeSuite+ 統合開発環境 ユーザーズマニュアル 起動編」を参照してください。

（2）プロジェクトの作成／読み込み

プロジェクトの新規作成（プロジェクトの種類，使用するデバイス，使用するビルド・ツールなどの定義），または既存のプロジェクトの読み込みを行います。

備考　“プロジェクトの作成／読み込み”についての詳細は，「CubeSuite+ 統合開発環境 ユーザーズマニュアル 起動編」を参照してください。

（3）周辺機能 パネルのオープン

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情報を設定するための周辺機能 パネルをオープンします。

（4）情報の設定

周辺機能 パネルで周辺機能（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情報を設定します。

（5）ソース・コードの確認

周辺機能 パネルで設定した情報に応じたソース・コード（デバイス・ドライバ・プログラム）を確認します。

（6）ソース・コードの出力

ソース・コード（デバイス・ドライバ・プログラム）を指定されたフォルダに出力します。

（7）レポート・ファイルの出力

レポート・ファイル（コード生成を用いて設定した情報を保持したファイル，ソース・コードに関する情報を保持したファイル）を指定されたフォルダに出力します。

（8）プロジェクトの保存

プロジェクトの保存を行います。

備考　“プロジェクトの保存”についての詳細は，「CubeSuite+ 統合開発環境 ユーザーズマニュアル 起動編」を参照してください。

3. 2　周辺機能 パネルのオープン

デバイス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情報を設定するための周辺機能 パネルをオープンします。

なお，周辺機能 パネルのオープンは，プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［コード生成（設計ツール）］→［周辺機能］（→周辺機能ノード）をダブルクリックすることにより行います。

図 3—1　周辺機能 パネルのオープン

備考　コード生成が未対応のデバイスがプロジェクトで定義された場合，プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］に“［コード生成（設計ツール）］ノード”は表示されません。

3. 3 情報の設定

「3. 2 周辺機能 パネルのオープン」でオープンした周辺機能 パネルの情報設定エリアで周辺機能（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情報を設定します。

備考 複数の周辺機能を制御する場合は，「3. 2 周辺機能 パネルのオープン」から「3. 3 情報の設定」の操作を繰り返し行うことになります。

3. 3. 1 入力規約

以下に，周辺機能 パネルに各種情報を設定する際の入力規約を示します。

（1）文字セット

以下に，コード生成が入力を許可している文字セットを示します。

表 3—1 文字セットの一覧

文字セット	概要
ASCII	半角のアルファベット（英字），半角の数字，半角の記号
Shift-JIS	全角のアルファベット（英字），全角の数字，全角の記号，全角のひらがな，全角のカタカナ，全角の漢字，および半角のカタカナ
EUC-JP	全角のアルファベット（英字），全角の数字，全角の記号，全角のひらがな，全角のカタカナ，全角の漢字，および半角のカタカナ
UTF-8	全角のアルファベット（英字），全角の数字，全角の記号，全角のひらがな，全角のカタカナ，全角の漢字（中国語を含む），および半角のカタカナ

（2）数値

以下に，コード生成が入力を許可している進数を示します。

表 3—2 進数の一覧

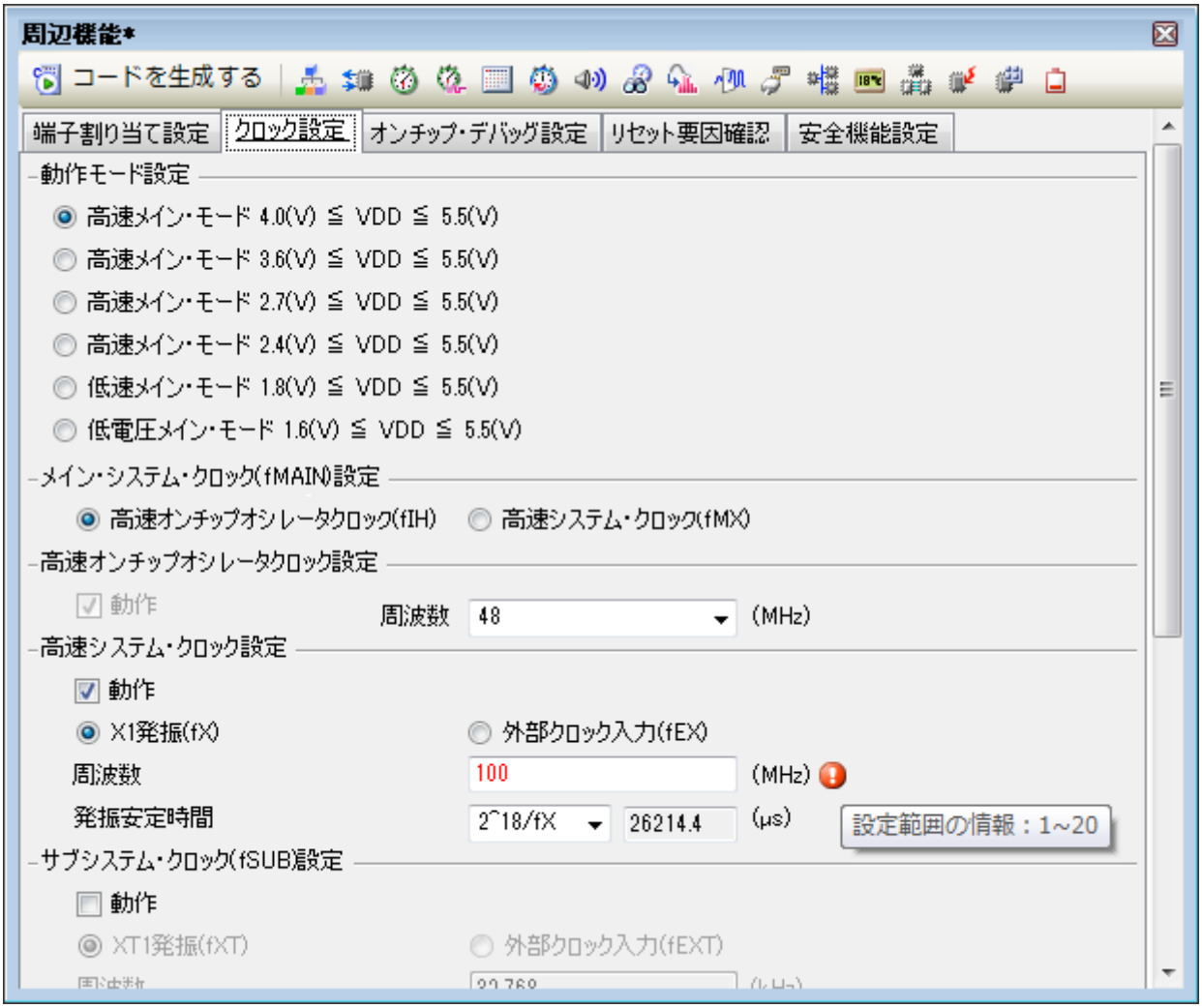
進数表記	概要
10 進数	1 ～ 9 の数字で始まり 0 ～ 9 の数字が続く数値，および 0
16 進数	0x で始まり 0 ～ 9 の数字，および a ～ f の英字が続く数値 （英字の大文字／小文字については，不問）

3. 3. 2　入力不備箇所に対するアイコン表示

コード生成では，周辺機能 パネルで不正な文字列が入力された際，および入力が必須な箇所に値が未入力の際，設定すべき情報として誤っていることを示すアイコンを該当箇所に表示するとともに，文字列を赤色表示し，入力の不備を警告します。

備考　アイコン上にマウス・カーソルを移動した際には，入力すべき文字列に関する情報（入力の不備を解決するためのヒント）がポップアップ表示されます。

図 3—2　入力不備箇所に対するアイコン表示

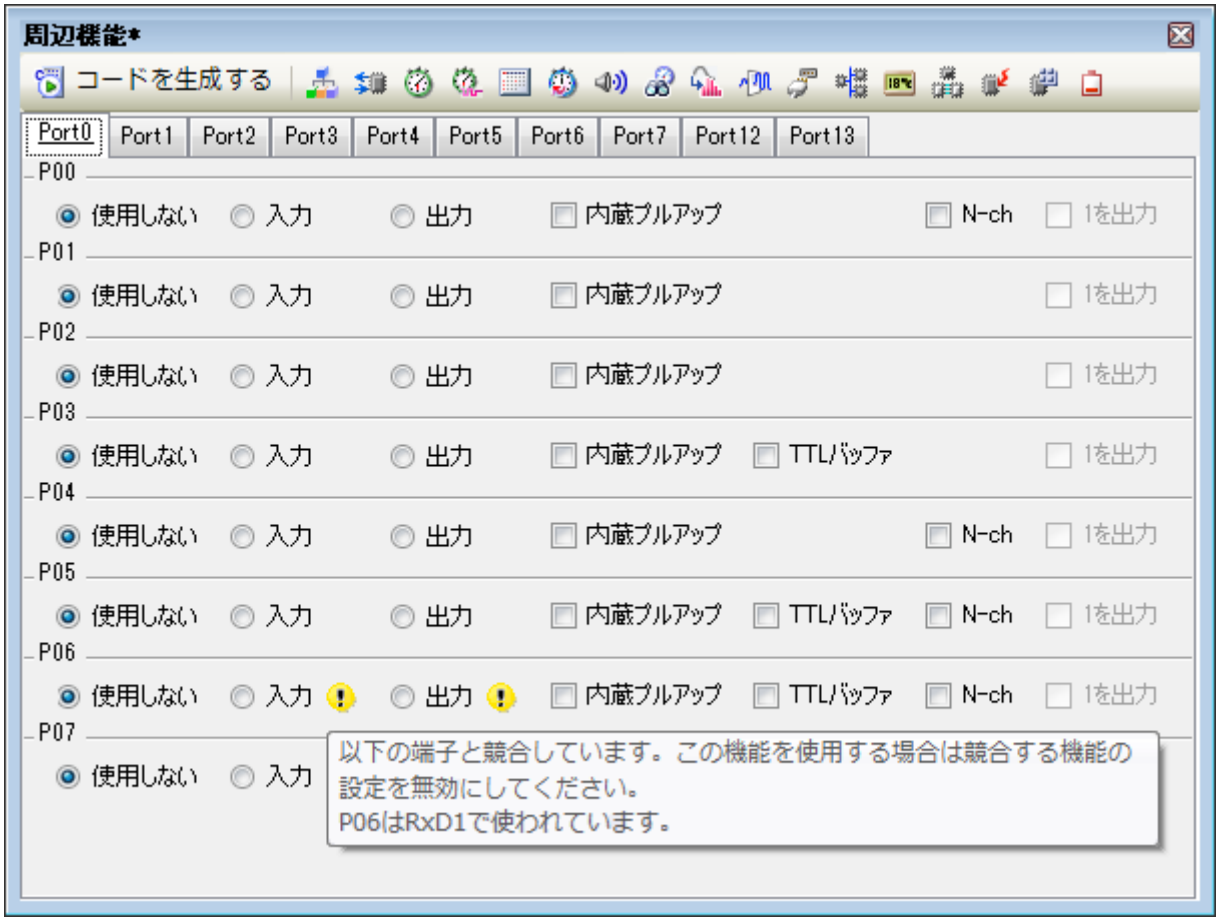


3. 3. 3　端子の競合に対するアイコン表示

コード生成では，周辺機能 パネルにおける各種周辺機能の設定に伴い，端子の競合が発生する項目に対しては，競合が発生することを示すアイコンを該当箇所に表示し，端子の競合を警告します。

備考　アイコン上にマウス・カーソルを移動した際には，端子の競合に関する情報（競合を回避するためのヒント）がポップアップ表示されます。

図 3—3　端子の競合に対するアイコン表示

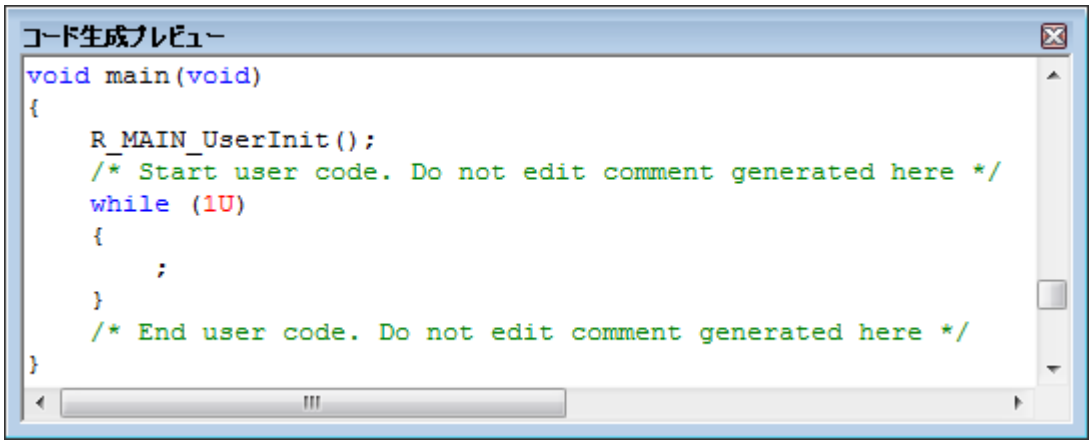


3.4　ソース・コードの確認

「3.3　情報の設定」で設定した情報に応じたソース・コード（デバイス・ドライバ・プログラム）を確認します。

なお，ソース・コードの確認は，プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［コード生成（設計ツール）］→ソース・コード・ノード（→ API 関数ノード）をダブルクリックすることによりオープンするコード生成プレビュー パネルで行います。

図 3—4　ソース・コードの確認



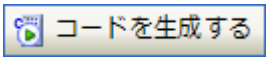
備考 1. プロジェクト・ツリー パネルのソース・ファイル名，または API 関数名を選択することにより，ソース・コードの表示を切り替えることができます。

2. コード生成プレビュー パネルに表示されるソース・コードの文字色は，以下の意味を持ちます。

表 3—3　ソース・コードの文字色

表示色	概要
緑	コメント文
青	C コンパイラの予約語
赤	数値
黒	コード部
グレー	ファイル名

3. コード生成プレビュー パネル内でソース・コードを編集することはできません。

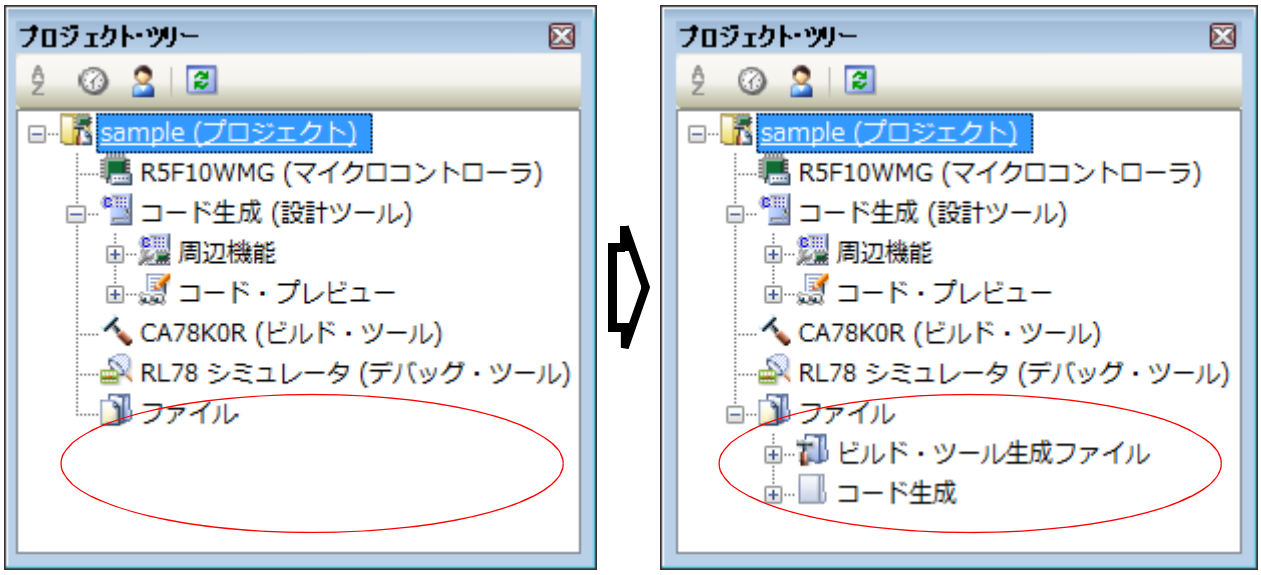
4. 一部の API 関数については，ソース・コードの出力時（周辺機能 パネルの 

ボタンをクリックした際）にレジスタ値などが計算され確定するものがあります。このため，本パネルに表示されるソース・コードは，実際に出力されるソース・コードと一致しない場合があります。

3.5　ソース・コードの出力

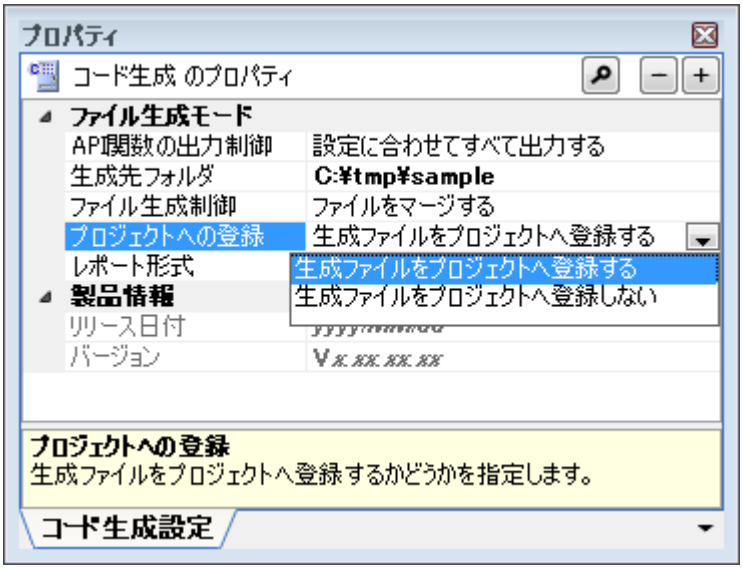
周辺機能 パネルの コードを生成する ボタンをクリックし，ソース・コード（デバイス・ドライバ・プログラム）を出力します。なお，ソース・コードの出力先は，プロパティ パネルの［コード生成設定］タブ→［ファイル生成モード］→［生成先フォルダ］で指定されたフォルダとなります。

図 3—5　ソース・コードの出力



備考　コードを生成する ボタンをクリックした際，ソース・コードを出力するとともに，該当ファイル群をプロジェクトに登録（プロジェクト・ツリー パネルに対する該当ソース・ファイル名の表示）する場合は，プロパティ パネルの［コード生成設定］タブ→［ファイル生成モード］→［プロジェクトへの登録］で“生成ファイルをプロジェクトへ登録する”を指定する必要があります。

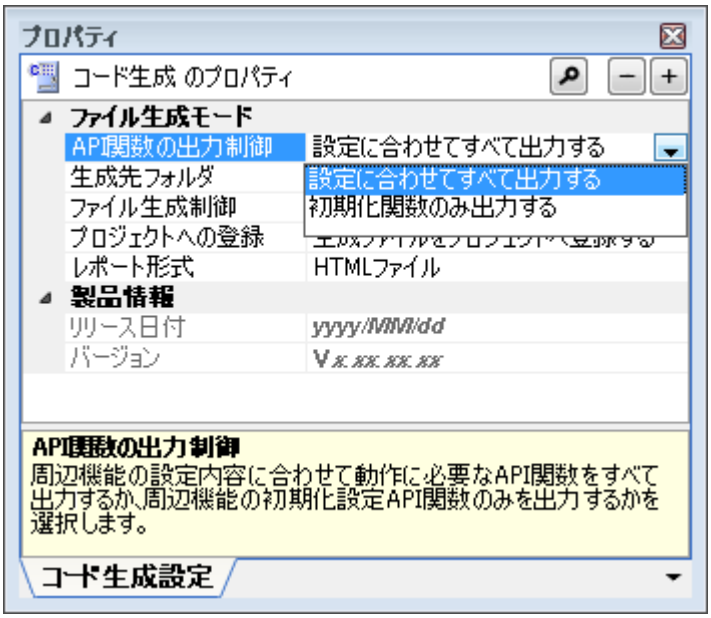
図 3—6　登録有無の設定



3. 5. 1 出力有無の設定

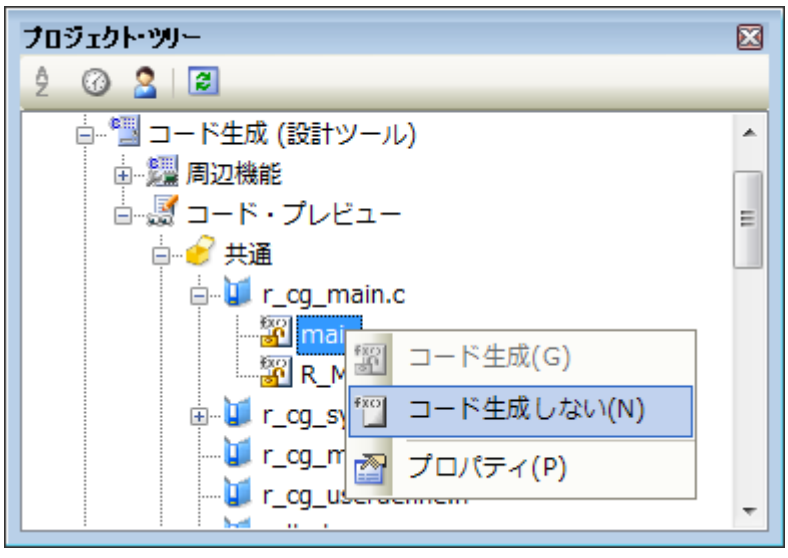
コード生成では，プロパティ パネルの［コード生成設定］タブ→［ファイル生成モード］→［API 関数の出力制御］で“設定に合わせてすべて出力する／初期化関数のみ出力する”を選択することにより，出力する API 関数の種類（全 API 関数，初期化用 API 関数のみ）を設定することができます。

図 3—7 出力有無の設定



コード生成では，プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［コード生成（設計ツール）］→［コード・プレビュー］→周辺機能ノード→ソース・コード・ノード→ API 関数ノードを選択したのち，マウスを右クリックすることにより表示されるコンテキスト・メニューから“コード生成／コードを生成しない”を選択することにより，API 関数単位での“該当ソース・コードの出力有無”についても設定することができます。

図 3—8 出力有無の設定



備考 出力有無の設定状況については，各 API 関数ノードの直前に表示されているアイコン種別により確認することができます。

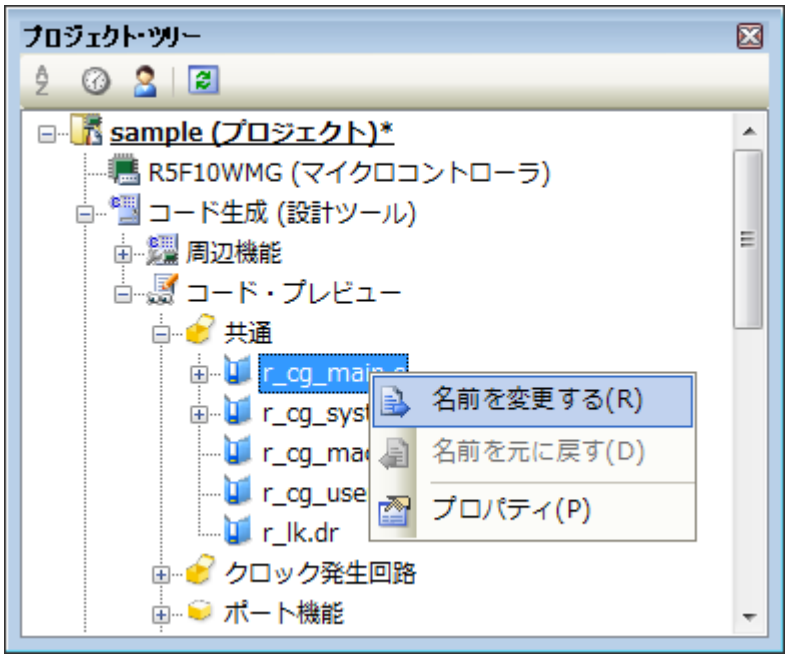
表 3—4 ソース・コードの出力有無

アイコン種別	概要
	該当 API 関数のソース・コードが出力されます。 なお，本アイコンが表示されている API 関数は，ソース・コードの出力が必須（  への変更不可）となります。
	該当 API 関数のソース・コードが出力されます。
	該当 API 関数のソース・コードが出力されません。

3. 5. 2 ファイル名の変更

コード生成では，プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［コード生成（設計ツール）］→［コード・プレビュー］→周辺機能ノード→ソース・コード・ノードを選択したのち，マウスを右クリックすることにより表示されるコンテキスト・メニューから“名前を変更する”を選択することにより，ファイル名を変更することができます。

図 3—9 ファイル名の変更

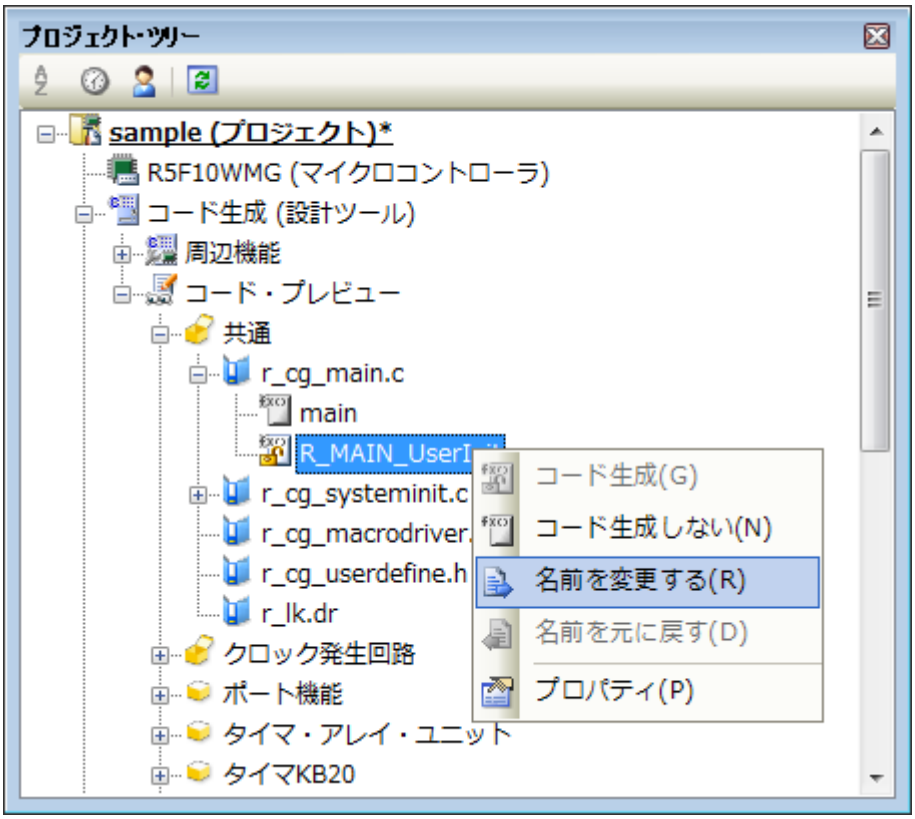


備考 コード生成が規定しているデフォルト・ファイル名に戻す際には，コンテキスト・メニューから“名前を元に戻す”を選択します。

3. 5. 3 API 関数名の変更

コード生成では，プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［コード生成（設計ツール）］→［コード・プレビュー］→周辺機能ノード→ソース・コード・ノード→ API 関数ノードを選択したのち，マウスを右クリックすることにより表示されるコンテキスト・メニューから“名前を変更する”を選択することにより，API 関数名を変更することができます。

図 3—10 API 関数名の変更



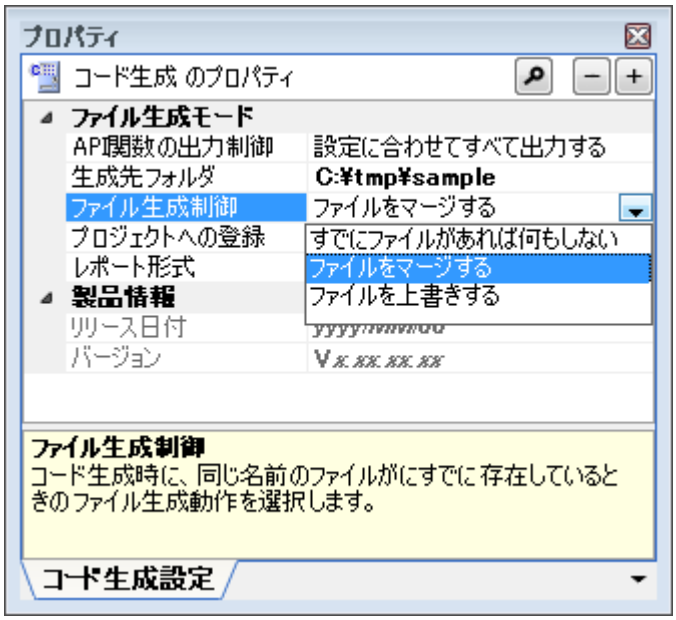
備考 1. コード生成が規定しているデフォルト API 関数名に戻す際には，コンテキスト・メニューから“名前を元に戻す”を選択します。

2. API 関数の中には，API 関数名を変更できないもの（main など）もあります。

3. 5. 4 出力モードの変更

コード生成では，プロパティ パネルの［コード生成設定］タブ→［ファイル生成モード］→［ファイル生成制御］でソース・コードの出力モード（すでにファイルがあれば何もしない，ファイルをマージする，ファイルを上書きする）を変更することができます。

図 3—11 出力モードの変更



出力モードの選択は，以下の 3 種類から行います。

表 3—5 ソース・コードの出力モード

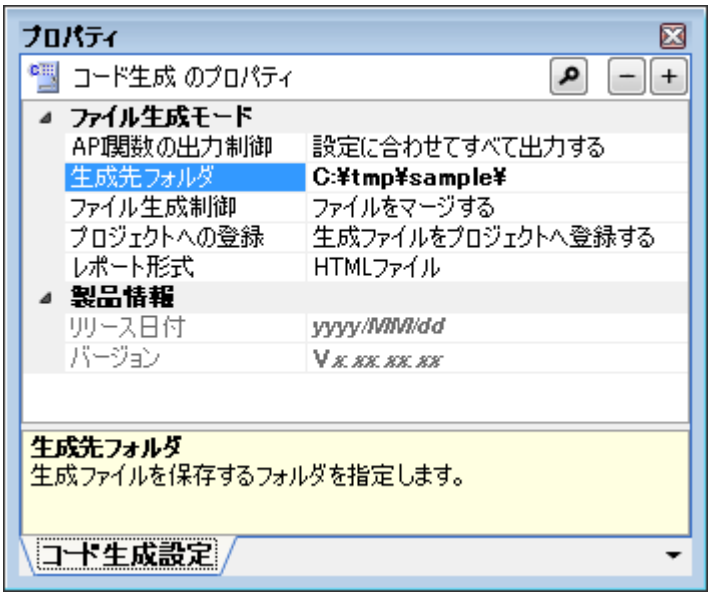
出力モード	概要
すでにファイルがあれば何もしない	同一のファイル名を有するファイルが存在していた場合，該当ファイルの出力を行いません。
ファイルをマージする	同一のファイル名を有するファイルが存在していた場合，該当ファイルをマージします。 なお，マージする部位については， /* Start user code …. Do not edit comment generated here */ から /* End user code. Do not edit comment generated here */ で囲まれた部位に限られます。
ファイルを上書きする	同一のファイル名を有するファイルが存在していた場合，該当ファイルを上書きします。

備考 ［ファイルをマージする］を選択した場合，マージする部位内の“{”と“}”の数は一致させる必要があります。“{”と“}”の数が不一致な際は，正しいマージ処理が行われません。

3. 5. 5　出力先の変更

コード生成では，プロパティ パネルの［コード生成設定］タブ→［ファイル生成モード］→［生成先フォルダ］でソース・コードの出力先を変更することができます。

図 3—12　出力先の変更



3. 6 レポート・ファイルの出力

周辺機能 パネル，またはコード生成プレビュー パネルをアクティブな状態にしたのち，［ファイル］メニュー→［コード生成レポート を保存］を選択し，レポート・ファイル（コード生成を用いて設定した情報を保持したファイル，ソース・コードに関する情報を保持したファイル）を出力します。

なお，レポート・ファイルの出力先は，プロパティ パネルの［コード生成設定］タブ→［ファイル生成モード］→［生成先フォルダ］で指定されたフォルダとなります。

備考 1. レポート・ファイルのファイル名は，“Function”，および“Macro”に規定されています。
なお，出力形式についての詳細は，「3. 6. 1 出力形式の変更」を参照してください。

表 3—6 レポート・ファイルの出力

ファイル名	概要
Function.*xxx*	ソース・コードに関する情報を保持したファイル
Macro.*xxx*	コード生成を用いて設定した情報を保持したファイル

2. レポート・ファイルの出力モードは，“ファイルを上書きする”に規定されています。

図 3—13 レポート・ファイル Function の出力例（HTML ファイル）

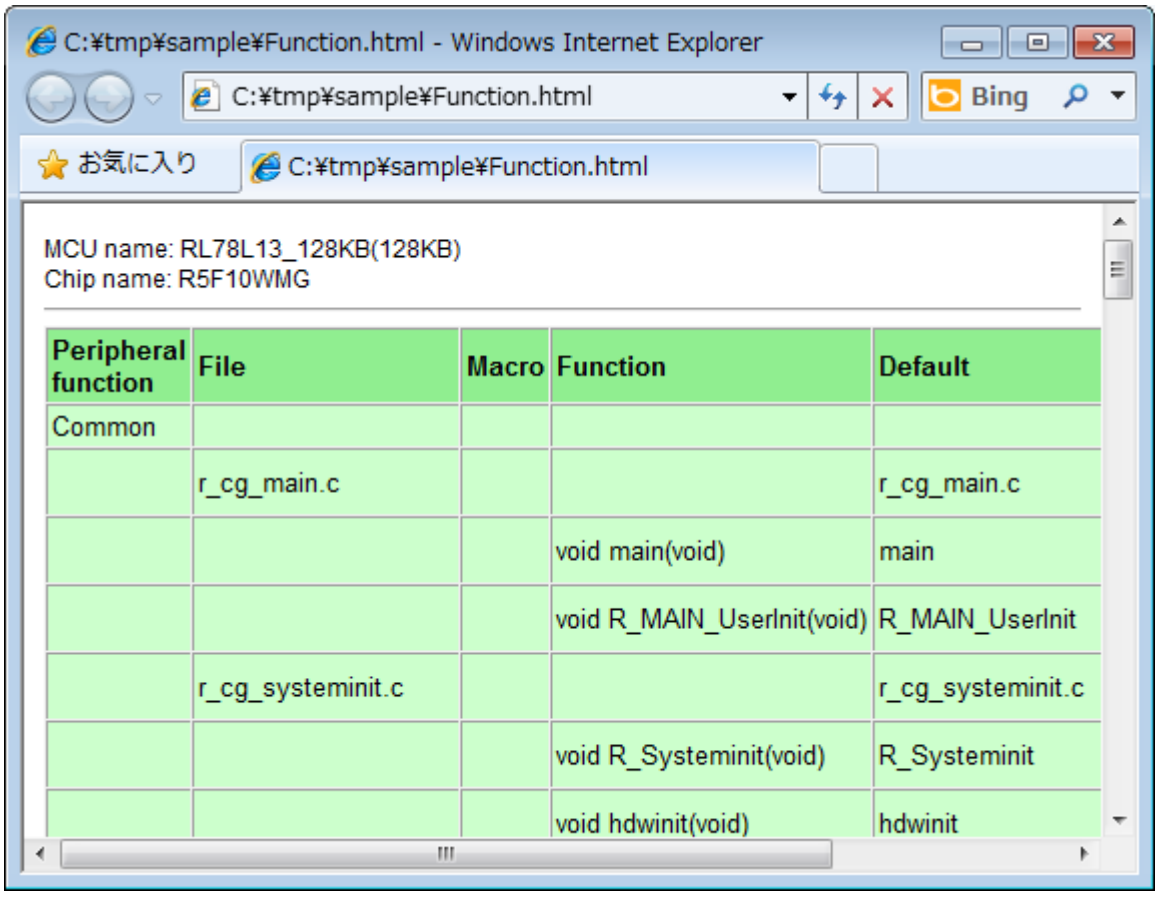
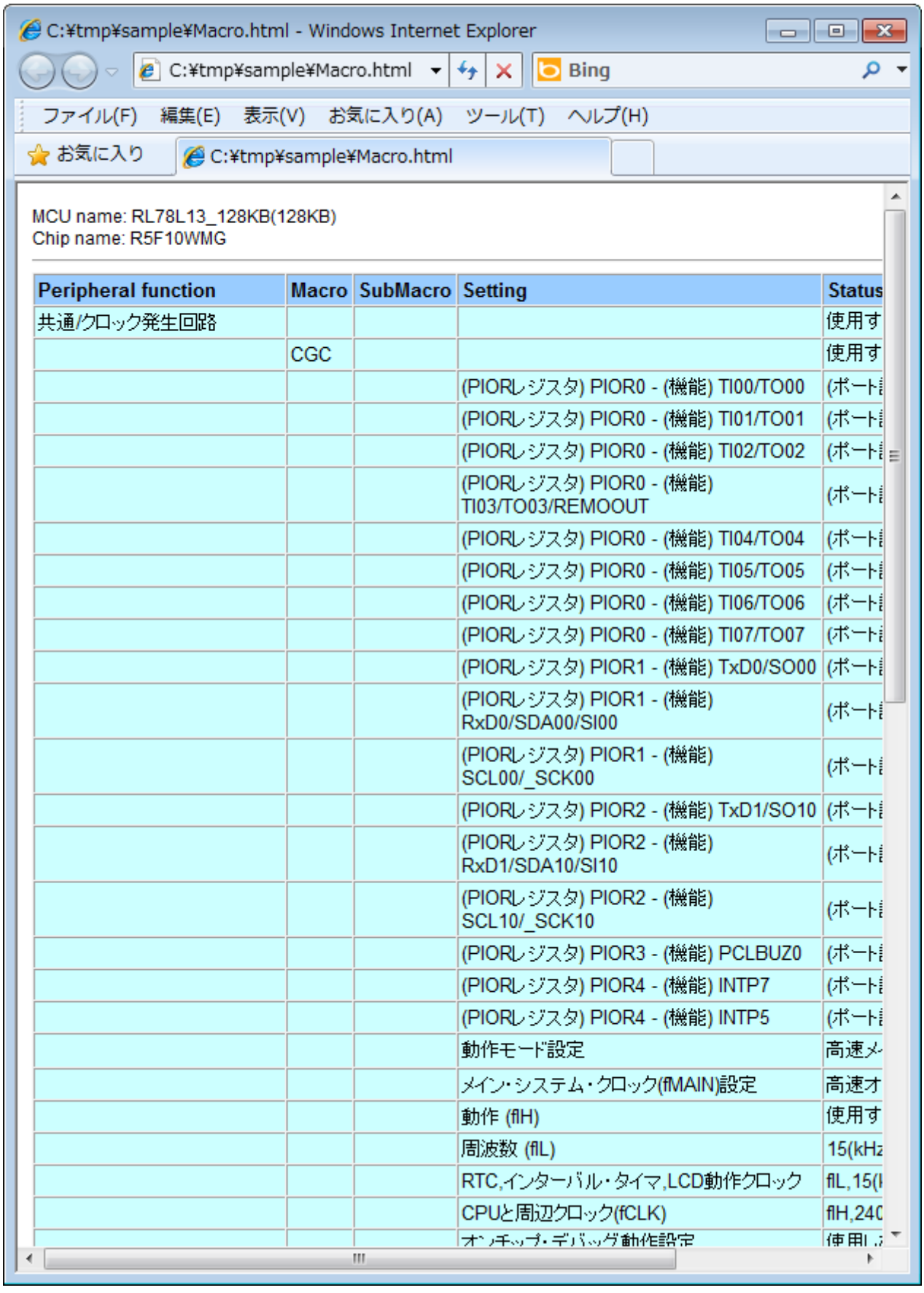


図 3—14 レポート・ファイル Macro の出力例（HTML ファイル）

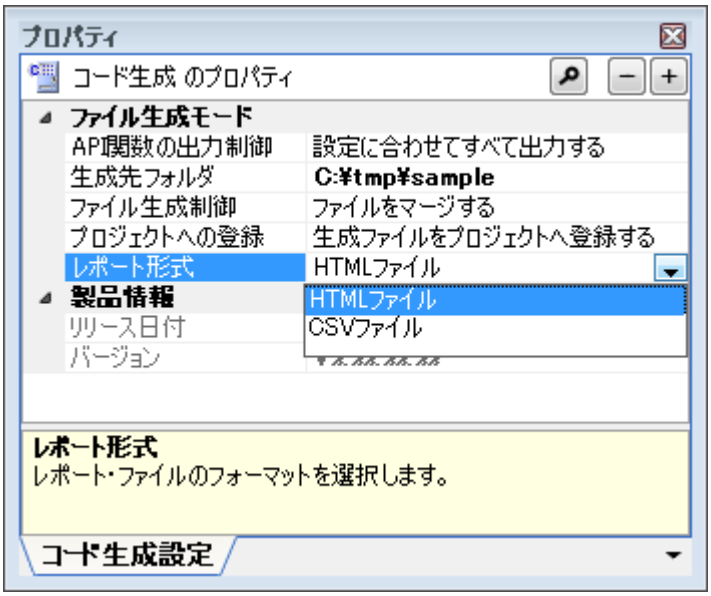


Peripheral function	Macro	SubMacro	Setting	Status
共通/クロック発生回路				使用す
	CGC			使用す
			(PIORレジスタ) PIOR0 - (機能) TI00/TO00	(ポート
			(PIORレジスタ) PIOR0 - (機能) TI01/TO01	(ポート
			(PIORレジスタ) PIOR0 - (機能) TI02/TO02	(ポート
			(PIORレジスタ) PIOR0 - (機能) TI03/TO03/REMOOUT	(ポート
			(PIORレジスタ) PIOR0 - (機能) TI04/TO04	(ポート
			(PIORレジスタ) PIOR0 - (機能) TI05/TO05	(ポート
			(PIORレジスタ) PIOR0 - (機能) TI06/TO06	(ポート
			(PIORレジスタ) PIOR0 - (機能) TI07/TO07	(ポート
			(PIORレジスタ) PIOR1 - (機能) TxD0/SO00	(ポート
			(PIORレジスタ) PIOR1 - (機能) RxD0/SDA00/SI00	(ポート
			(PIORレジスタ) PIOR1 - (機能) SCL00/_SCK00	(ポート
			(PIORレジスタ) PIOR2 - (機能) TxD1/SO10	(ポート
			(PIORレジスタ) PIOR2 - (機能) RxD1/SDA10/SI10	(ポート
			(PIORレジスタ) PIOR2 - (機能) SCL10/_SCK10	(ポート
			(PIORレジスタ) PIOR3 - (機能) PCLBUZ0	(ポート
			(PIORレジスタ) PIOR4 - (機能) INTP7	(ポート
			(PIORレジスタ) PIOR4 - (機能) INTP5	(ポート
			動作モード設定	高速メ
			メイン・システム・クロック(fMAIN)設定	高速オ
			動作 (fIH)	使用す
			周波数 (fIL)	15(kHz
			RTC,インターバル・タイマ,LCD動作クロック	fIL,15(
			CPUと周辺クロック(fCLK)	fIH,240
			オンチップ・デバッグ動作設定	使用し

3. 6. 1 出力形式の変更

コード生成では，プロパティ パネルの［コード生成設定］タブ→［ファイル生成モード］→［レポート形式］でレポート・ファイルの出力形式（HTML ファイル，CSV ファイル）を変更することができます。

図 3—15 出力形式の変更



備考 レポート・ファイルの出力形式は，以下の 2 種類から選択します。

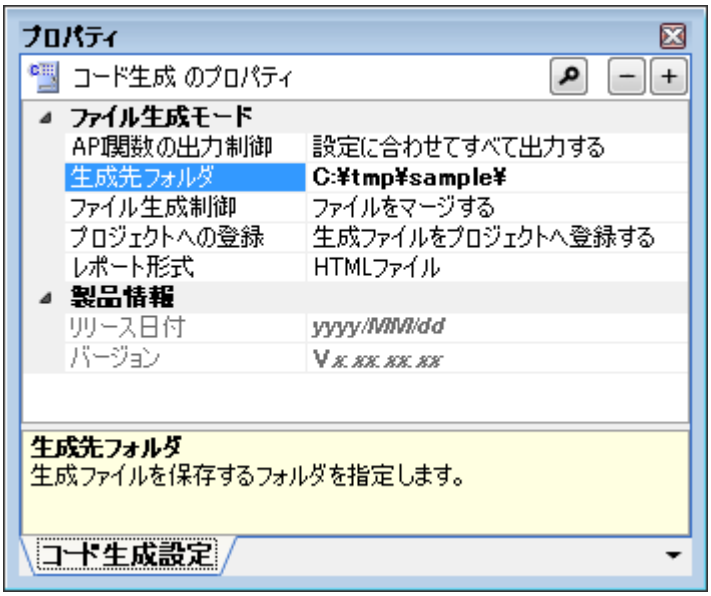
表 3—7 レポート・ファイルの出力形式

出力形式	概要
HTML ファイル	HTML 形式で出力します。
CSV ファイル	CSV 形式で出力します。

3. 6. 2　出力先の変更

コード生成では，プロパティ パネルの［コード生成設定］タブ→［ファイル生成モード］→［生成先フォルダ］でレポート・ファイルの出力先を変更することができます。

図 3—16　出力先の変更



付録 A　ウインドウ・リファレンス

本付録では，設計ツールのウインドウ／パネル／ダイアログについて説明します。

A. 1　説　　明

以下に，設計ツールのウインドウ／パネル／ダイアログの一覧を示します。

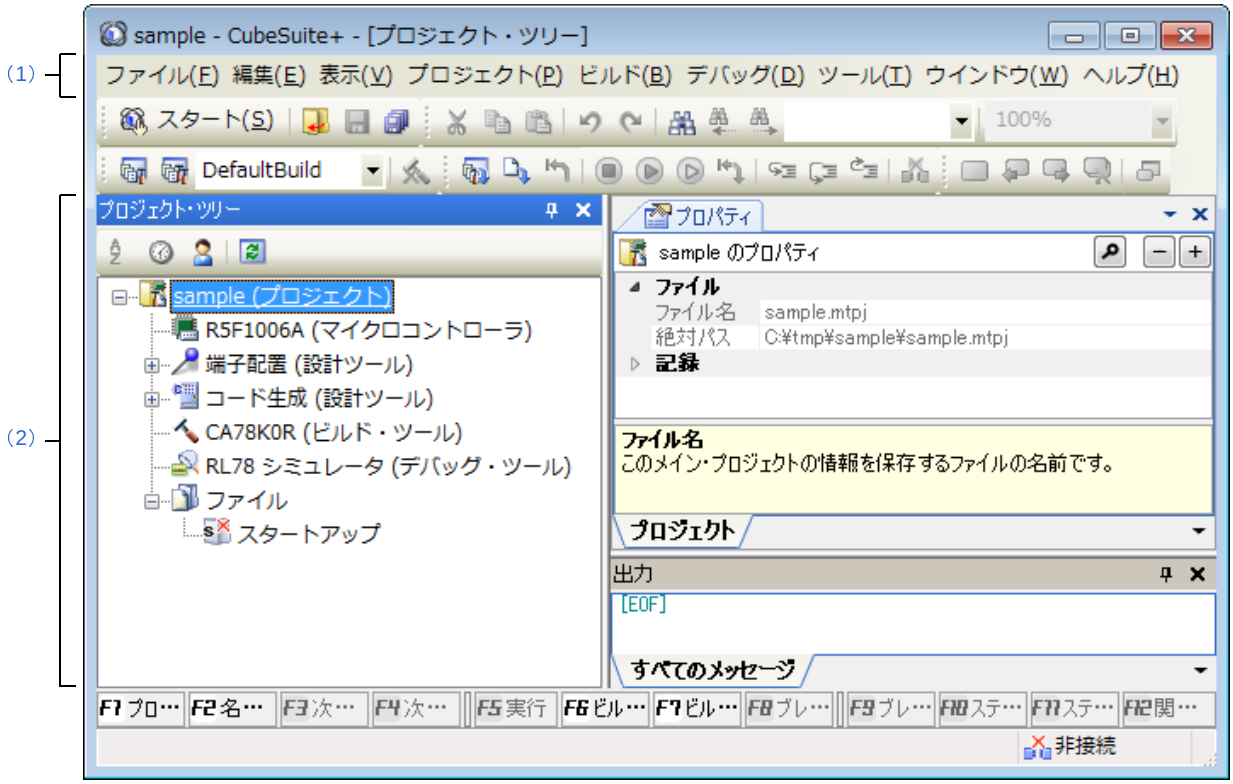
表 A—1　ウインドウ／パネル／ダイアログの一覧

ウインドウ／パネル／ダイアログ名	機能概要
メイン・ウインドウ	CubeSuite+ を起動した際，最初にオープンするウインドウであり，本ウインドウから CubeSuite+ が提供している各種コンポーネント（設計ツール，ビルド・ツールなど）に対する操作を行います。
プロジェクト・ツリー パネル	プロジェクトの構成要素（マイクロコントローラ，設計ツール，ビルド・ツールなど）をツリー形式で表示します。
プロパティ パネル	プロジェクト・ツリー パネルで選択したノードに対応した情報の表示，および設定の変更を行います。
端子配置表 パネル	マイクロコントローラの各端子に関する情報を記述します。
端子配置図 パネル	端子配置表 パネルにおける情報の記述状況を表示します。
周辺機能 パネル	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情報を設定します。
コード生成プレビュー パネル	周辺機能 パネルの設定内容に応じたソース・コードを確認します。
出力 パネル	CubeSuite+ が提供している各種コンポーネント（設計ツール，ビルド・ツールなど）の操作ログを表示します。
列の選択 ダイアログ	本ダイアログに表示されている項目を端子配置表に表示するか否かの選択，および端子配置表に対する列の追加／削除を行います。
新しい列 ダイアログ	端子配置表に列を追加します。
名前を付けて保存 ダイアログ	ファイルに名前を付けて保存します。

メイン・ウインドウ

CubeSuite+ を起動した際，最初にオープンするウインドウであり，本ウインドウから CubeSuite+ が提供している各種コンポーネント（設計ツール，ビルド・ツールなど）に対する操作を行います。

図 A―1　メイン・ウインドウ



ここでは，次の項目について説明します。

- ［オープン方法］
- ［各エリアの説明］

［オープン方法］

- Windows の［スタート］メニューから［プログラム］→［Renesas Electronics CubeSuite+］→［CubeSuite+］を選択

［各エリアの説明］

（1）メニューバー

本エリアは，以下に示したメニュー群から構成されています。

(a)［ファイル］メニュー

端子配置表 を保存	端子配置表 パネル専用部分 レポート・ファイル（端子配置を用いて設定した情報を保持したファイル：端子配置表）を既存のファイルに上書きします。
名前を付けて 端子配置表 を保存 ...	端子配置表 パネル専用部分 レポート・ファイル（端子配置を用いて設定した情報を保持したファイル：端子配置表）に名前を付けて保存するための名前を付けて保存 ダイアログをオープンします。
端子配置図 を保存	端子配置図 パネル専用部分 レポート・ファイル（端子配置を用いて設定した情報を保持したファイル：端子配置図）を既存のファイルに上書きします。
名前を付けて 端子配置図 を保存 ...	端子配置図 パネル専用部分 レポート・ファイル（端子配置を用いて設定した情報を保持したファイル：端子配置図）に名前を付けて保存するための名前を付けて保存 ダイアログをオープンします。
コード生成レポート を保存	周辺機能 パネル／コード生成プレビュー パネル専用部分 レポート・ファイル（コード生成を用いて設定した情報を保持したファイル，ソース・コードに関する情報を保持したファイル）を出力します。 - レポート・ファイルの出力形式は，プロパティ パネルの［コード生成設定］タブ→［ファイル生成モード］→［レポート形式］で選択された形式（HTML ファイル，または CSV ファイル）となります。 - レポート・ファイルの出力先は，プロパティ パネルの［コード生成設定］タブ→［ファイル生成モード］→［生成先フォルダ］で指定されたフォルダとなります。
出力 - *タブ名* を保存	出力 パネル専用部分 該当タブのメッセージを既存のファイルに上書き保存します。
名前を付けて 出力 - *タブ名* を保存 ...	出力 パネル専用部分 該当タブのメッセージに名前を付けて保存するための名前を付けて保存 ダイアログをオープンします。

(b)［編集］メニュー

元に戻す	プロパティ パネル専用部分 直前に行った編集作業を取り消します。
切り取り	プロパティ パネル専用部分 選択している文字列を切り取り，クリップ・ボードに保存します。
コピー	プロパティ パネル／出力 パネル専用部分 選択している文字列をクリップ・ボードに保存します。
貼り付け	プロパティ パネル専用部分 指定された箇所に，クリップ・ボードの内容を挿入します。
削除	プロパティ パネル専用部分 選択している文字列を削除します。

すべて選択	プロパティ パネル／出力 パネル専用部分 編集中の項目に表示されている全文字列，またはメッセージ・エリアに表示されている全文字列を選択します。
検索 ...	端子配置表 パネル／コード生成プレビュー パネル／出力 パネル専用部分 文字列検索を行うための検索・置換 ダイアログを［クイック検索］タブが選択 された状態でオープンします。
置換 ...	出力 パネル専用部分 文字列置換を行うための検索・置換 ダイアログを［一括置換］タブが選択された状態でオープンします。

(c)［表示］メニュー

プロジェクト・ツリー	プロジェクト・ツリー パネル専用部分 プロジェクト・ツリー パネルをオープンします。	
プロパティ	プロパティ パネル専用部分 プロパティ パネルをオープンします。	
出力	出力 パネル専用部分 出力 パネルをオープンします。	
端子配置	次のカスケード・メニューを表示します。	
	端子配置表	端子配置表 パネル専用部分 端子配置表 パネルをオープンします。
	端子配置図	端子配置図 パネル専用部分 端子配置図 パネルをオープンします。
コード生成 2	次のカスケード・メニューを表示します。	
	周辺機能	周辺機能 パネル専用部分 周辺機能 パネルをオープンします。
	コード生成プレビュー	コード生成プレビュー パネル専用部分 コード生成プレビュー パネルをオープンします。

(d)［ヘルプ］メニュー

プロジェクト・ツリー パネルのヘルプを開く	プロジェクト・ツリー パネル専用部分 プロジェクト・ツリー パネルのヘルプを表示します。
プロパティ パネルのヘルプを開く	プロパティ パネル専用部分 プロパティ パネルのヘルプを表示します。
端子配置表 パネルのヘルプを開く	端子配置表 パネル専用部分 端子配置表 パネルのヘルプを表示します。
端子配置図 パネルのヘルプを開く	端子配置図 パネル専用部分 端子配置図 パネルのヘルプを表示します。
コード生成 パネルのヘルプを開く	周辺機能 パネル専用部分 周辺機能 パネルのヘルプを表示します。
コード生成プレビュー パネルのヘルプを開く	コード生成プレビュー パネル専用部分 コード生成プレビュー パネルのヘルプを表示します。

出力 パネルのヘルプを開く	出力 パネル専用部分 出力 パネルのヘルプを表示します。

（2）パネル表示エリア

本エリアは，用途別に用意された各種パネルから構成されています。

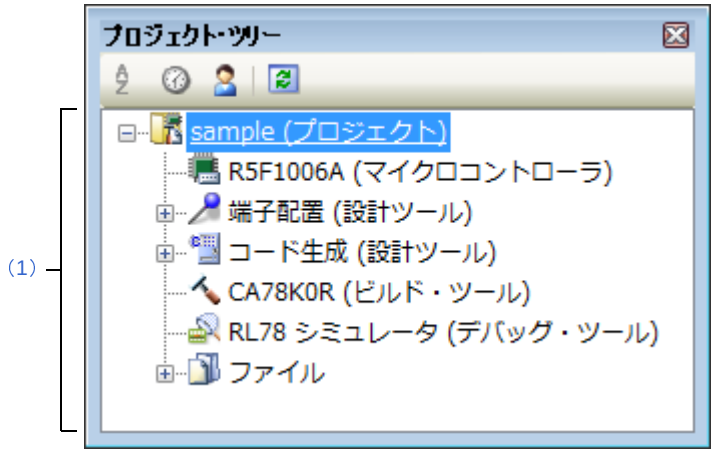
本エリアについての詳細は，以下を参照してください。

- プロジェクト・ツリー パネル
- プロパティ パネル
- 端子配置表 パネル
- 端子配置図 パネル
- 周辺機能 パネル
- コード生成プレビュー パネル
- 出力 パネル

プロジェクト・ツリー パネル

プロジェクトの構成要素（マイクロコントローラ，設計ツール，ビルド・ツールなど）をツリー形式で表示します。

図 A―2　プロジェクト・ツリー パネル



ここでは，次の項目について説明します。

- ［オープン方法］
- ［各エリアの説明］
- ［コンテキスト・メニュー］

［オープン方法］

- ［表示］メニュー→［プロジェクト・ツリー］を選択

［各エリアの説明］

(1) プロジェクト・ツリー・エリア

プロジェクトの構成要素（マイクロコントローラ，設計ツール，ビルド・ツールなど）をツリー形式で表示します。

(a) 端子配置（設計ツール）

本ノードは，以下に示した端子配置ノードから構成されています。

端子配置表	マイクロコントローラの各端子に関する情報を記述するための端子配置表 パネルをオープンします。
端子配置図	端子配置表 パネルにおける情報の記述状況を表示するための端子配置図 パネルをオープンします。

（b）コード生成（設計ツール）

本ノードは，［周辺機能］，［コード・プレビュー］から構成されています。

- ［周辺機能］

本ノードは，対象デバイスがサポートしている周辺機能（クロック発生回路，ポート機能など）に対応した周辺機能ノードから構成されています。

周辺機能ノード	周辺機能ノードをダブルクリック，または周辺機能ノードを選択したのち，［Enter］キーを押下することにより，該当周辺機能を制御するうえで必要な情報を設定するための周辺機能 パネルをオープンします。

各周辺機能ノードの直前に表示されているアイコンは，以下の意味を持ちます。

	該当周辺機能 パネルに対する操作を実施済み。
	該当周辺機能 パネルに対する操作を未実施。
 ， 	他の周辺機能ノードに対する操作の影響を受け，設定内容に問題が発生。

- ［コード・プレビュー］

本ノードは，対象デバイスがサポートしている周辺機能（クロック発生回路，ポート機能など）に対応した周辺機能ノードから構成されています。

周辺機能ノード	本ノードの下層に配置されているソース・コード・ノード／API 関数ノードをダブルクリック，またはソース・コード・ノード／API 関数ノードを選択したのち，［Enter］キーを押下することにより，周辺機能 パネルの設定内容に応じたソース・コードを確認するためのコード生成 プレビュー パネルをオープンします。

各周辺機能ノードの直前に表示されているアイコンは，以下の意味を持ちます。

	該当周辺機能 パネルに対する操作を実施済み。
	該当周辺機能 パネルに対する操作を未実施。

［コンテキスト・メニュー］

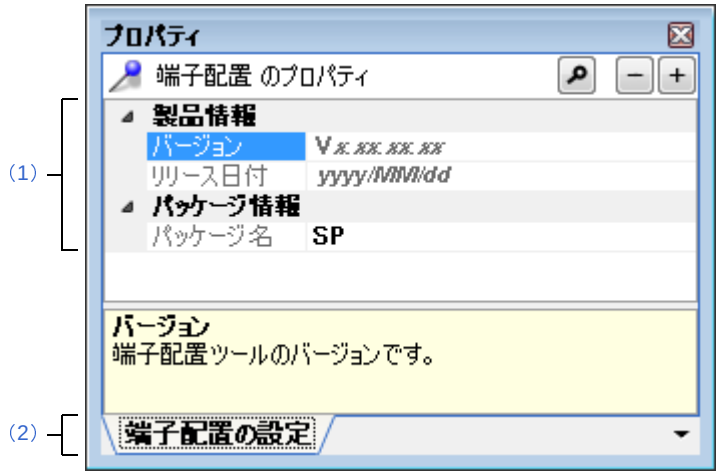
マウスを右クリックすることにより表示されるコンテキスト・メニューは，以下のとおりです。

リセット時の設定に戻す	選択しているノードの設定内容をデフォルトに戻します。
プロパティ	選択しているノードに対応したプロパティ パネルをオープンします。

プロパティ パネル

プロジェクト・ツリー パネルで選択したノードに対応した情報の表示，および設定の変更を行います。

図 A—3　プロパティ パネル



ここでは，次の項目について説明します。

- ［オープン方法］
- ［各エリアの説明］
- ［コンテキスト・メニュー］

［オープン方法］

- プロジェクト・ツリー パネルにおいて，［*Project name*（プロジェクト）］→［端子配置（設計ツール）］を選択したのち，［表示］メニュー→［プロパティ］を選択
- プロジェクト・ツリー パネルにおいて，［*Project name*（プロジェクト）］→［端子配置（設計ツール）］を選択したのち，コンテキスト・メニューから［プロパティ］を選択
- プロジェクト・ツリー パネルにおいて，［*Project name*（プロジェクト）］→［端子配置（設計ツール）］→［端子配置表］を選択したのち，［表示］メニュー→［プロパティ］を選択
- プロジェクト・ツリー パネルにおいて，［*Project name*（プロジェクト）］→［端子配置（設計ツール）］→［端子配置表］を選択したのち，コンテキスト・メニューから［プロパティ］を選択
- プロジェクト・ツリー パネルにおいて，［*Project name*（プロジェクト）］→［端子配置（設計ツール）］→［端子配置図］を選択したのち，［表示］メニュー→［プロパティ］を選択
- プロジェクト・ツリー パネルにおいて，［*Project name*（プロジェクト）］→［端子配置（設計ツール）］→［端子配置図］を選択したのち，コンテキスト・メニューから［プロパティ］を選択
- プロジェクト・ツリー パネルにおいて，［*Project name*（プロジェクト）］→［コード生成（設計ツール）］を選択したのち，［表示］メニュー→［プロパティ］を選択
- プロジェクト・ツリー パネルにおいて，［*Project name*（プロジェクト）］→［コード生成（設計ツール）］を選択したのち，コンテキスト・メニューから［プロパティ］を選択

- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［コード生成（設計ツール）］→［周辺機能］（→周辺機能ノード）を選択したのち，［表示］メニュー→［プロパティ］を選択
- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［コード生成（設計ツール）］→［周辺機能］（→周辺機能ノード）を選択したのち，コンテキスト・メニューから［プロパティ］を選択
- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［コード生成（設計ツール）］→［コード・プレビュー］（→周辺機能ノード→ソース・コード・ノード→ API 関数ノード）を選択したのち，［表示］メニュー→［プロパティ］を選択
- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［コード生成（設計ツール）］→［コード・プレビュー］（→周辺機能ノード→ソース・コード・ノード→ API 関数ノード）を選択したのち，コンテキスト・メニューから［プロパティ］を選択

備考 1. すでに本パネルがオープンしていた場合，プロジェクト・ツリー パネルの［端子配置（設計ツール）］を選択することにより，表示内容が切り替わります。

2. すでに本パネルがオープンしていた場合，プロジェクト・ツリー パネルの［端子配置表］を選択することにより，表示内容が切り替わります。

3. すでに本パネルがオープンしていた場合，プロジェクト・ツリー パネルの［端子配置図］を選択することにより，表示内容が切り替わります。

4. すでに本パネルがオープンしていた場合，プロジェクト・ツリー パネルの［コード生成（設計ツール）］を選択することにより，表示内容が切り替わります。

5. すでに本パネルがオープンしていた場合，プロジェクト・ツリー パネルの［周辺機能］（→周辺機能ノード）を選択することにより，表示内容が該当ノードに対応したものへと切り替わります。

6. すでに本パネルがオープンしていた場合，プロジェクト・ツリー パネルの［コード・プレビュー］（→周辺機能ノード→ソース・コード・ノード→ API 関数ノード）を選択することにより，表示内容が該当ノードに対応したものへと切り替わります。

［各エリアの説明］

（1）詳細情報表示／変更エリア

プロジェクト・ツリー パネルで選択したノードに対応した情報の表示，および設定の変更を行います。

なお，本エリアの表示内容については，プロジェクト・ツリー パネルで選択したノードの種類により異なります。

各カテゴリの直前に表示されている ⊞，および ⊟ は，以下の意味を持ちます。

⊞	カテゴリ内の項目が“折りたたみ表示”されていることを示します。
⊟	カテゴリ内の項目が“展開表示”されていることを示します。

備考 ⊞ と ⊟ の切り替えは，本マークのクリック，またはカテゴリ名のダブルクリックにより実現されます。

(2) タブ選択エリア

このパネルには，次のタブが存在します（各タブ上における表示内容／設定方法についての詳細は，該当するタブの項を参照してください）。

- ［端子配置の設定］タブ
- ［端子配置表の情報］タブ
- ［端子配置図の設定］タブ
- ［コード生成設定］タブ
- ［周辺機能情報］タブ（製品情報）
- ［周辺機能情報］タブ（周辺機能情報）
- ［コード・プレビュー情報］タブ（製品情報）
- ［コード・プレビュー情報］タブ（周辺機能情報）
- ［コード・プレビュー設定］タブ（ファイル情報）
- ［コード・プレビュー設定］タブ（関数情報）

[コンテキスト・メニュー]

マウスを右クリックすることにより表示されるコンテキスト・メニューは，以下のとおりです。

(1) 項目を編集中の場合

元に戻す	直前に行った編集作業を取り消します。
切り取り	選択している文字列を切り取り，クリップ・ボードに保存します。
コピー	選択している文字列をクリップ・ボードに保存します。
貼り付け	指定された箇所に，クリップ・ボードの内容を挿入します。
削除	選択している文字列を削除します。
すべて選択	編集中の項目に表示されている全文字列を選択します。

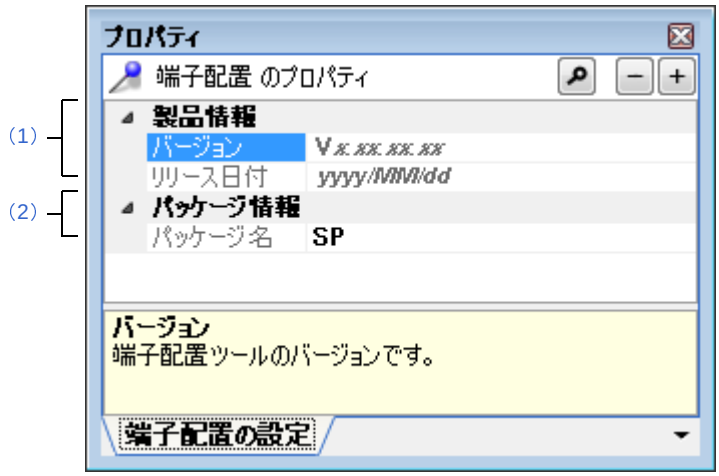
(2) 項目を編集中以外の場合

デフォルトに戻す	選択している項目の設定内容をデフォルトに戻します。
すべてデフォルトに戻す	本タブに表示されている全項目の設定内容をデフォルトに戻します。

［端子配置の設定］タブ

プロジェクト・ツリー パネルで選択した［端子配置（設計ツール）］に対応した情報（製品情報，パッケージ情報）の表示を行います。

図 A―4　［端子配置の設定］タブ



ここでは，次の項目について説明します。

- ［オープン方法］
- ［各エリアの説明］

［オープン方法］

- プロジェクト・ツリー パネルにおいて，［*Project name*（プロジェクト）］→［端子配置（設計ツール）］を選択したのち，［表示］メニュー→［プロパティ］を選択
- プロジェクト・ツリー パネルにおいて，［*Project name*（プロジェクト）］→［端子配置（設計ツール）］を選択したのち，コンテキスト・メニューから［プロパティ］を選択

備考　すでに本パネルがオープンしていた場合，プロジェクト・ツリー パネルの［端子配置（設計ツール）］を選択することにより，表示内容が切り替わります。

［各エリアの説明］

(1)［製品情報］カテゴリ

端子配置に関する製品情報（バージョン，リリース日付）の表示を行います。

バージョン	端子配置（端子配置プラグイン）のバージョンを表示します。
リリース日付	端子配置（端子配置プラグイン）のリリース日付を表示します。

(2)［パッケージ情報］カテゴリ

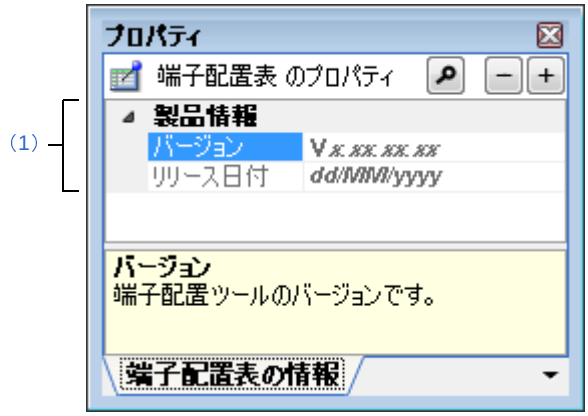
端子配置図 パネルに端子配置図として表示するマイクロコントローラの形状（パッケージ名）を選択します。

パッケージ名	端子配置図として表示するマイクロコントローラの形状を選択します。

［端子配置表の情報］タブ

プロジェクト・ツリー パネルで選択した［端子配置表］に対応した情報（製品情報）の表示を行います。

図 A―5 ［端子配置表の情報］タブ



ここでは，次の項目について説明します。

- ［オープン方法］
- ［各エリアの説明］

［オープン方法］

- プロジェクト・ツリー パネルにおいて，［*Project name*（プロジェクト）］→［端子配置（設計ツール）］→［端子配置表］を選択したのち，［表示］メニュー→［プロパティ］を選択
- プロジェクト・ツリー パネルにおいて，［*Project name*（プロジェクト）］→［端子配置（設計ツール）］→［端子配置表］を選択したのち，コンテキスト・メニューから［プロパティ］を選択

備考　すでに本パネルがオープンしていた場合，プロジェクト・ツリー パネルの［端子配置表］を選択することにより，表示内容が切り替わります。

［各エリアの説明］

(1)［製品情報］カテゴリ

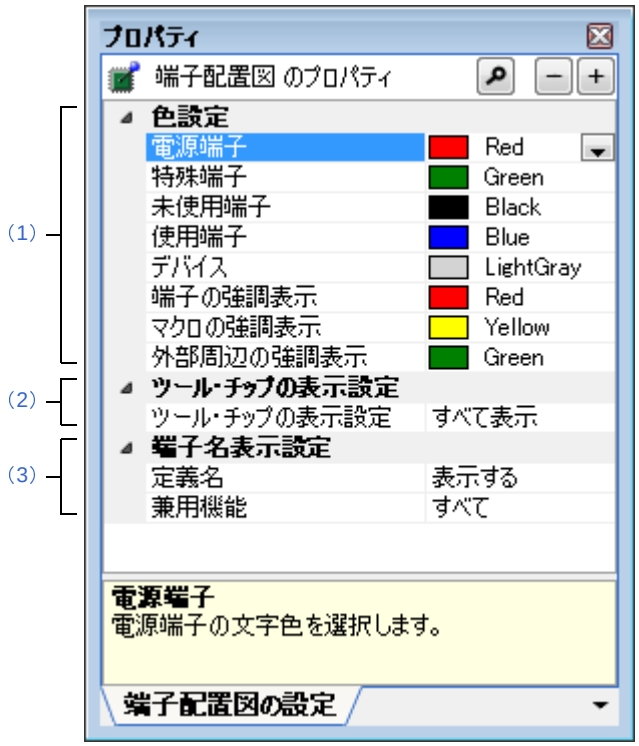
端子配置に関する製品情報（バージョン，リリース日付）の表示を行います。

バージョン	端子配置（端子配置プラグイン）のバージョンを表示します。
リリース日付	端子配置（端子配置プラグイン）のリリース日付を表示します。

［端子配置図の設定］タブ

プロジェクト・ツリー パネルで選択した［端子配置図］に対応した情報（色設定，ツール・チップの表示設定，端子名表示設定）の表示，および設定の変更を行います。

図 A―6 ［端子配置図の設定］タブ



ここでは，次の項目について説明します。

- ［オープン方法］
- ［各エリアの説明］

［オープン方法］

- プロジェクト・ツリー パネルにおいて，［*Project name*（プロジェクト）］→［端子配置（設計ツール）］→［端子配置図］を選択したのち，［表示］メニュー→［プロパティ］を選択
- プロジェクト・ツリー パネルにおいて，［*Project name*（プロジェクト）］→［端子配置（設計ツール）］→［端子配置図］を選択したのち，コンテキスト・メニューから［プロパティ］を選択

備考　すでに本パネルがオープンしていた場合，プロジェクト・ツリー パネルの［端子配置図］を選択することにより，表示内容が切り替わります。

［各エリアの説明］

(1)［色設定］カテゴリ

端子配置図の端子をグループ単位（電源端子，特殊端子など）に区別するための表示色を選択します。

電源端子	電源端子（用途が電源に限定されている端子）の表示色を選択します。
特殊端子	特殊端子（用途が規定されている端子）の表示色を選択します。
未使用端子	未使用端子（端子配置表 パネルにおいて，用途が未設定の兼用端子）の表示色を選択します。
使用端子	使用端子（端子配置表 パネルにおいて，用途が設定済みの兼用端子）の表示色を選択します。
デバイス	マイクロコントローラ本体部の表示色を選択します。
端子の強調表示	端子配置表 パネルの［端子番号］タブで選択された項目に対応した端子の背景色を選択します。
マクロの強調表示	端子配置表 パネルの［マクロ］タブで選択された項目に対応した端子の背景色を選択します。
外部周辺の強調表示	端子配置表 パネルの［外部周辺］タブで選択された項目に対応した端子の背景色を選択します。

(2)［ツール・チップの表示設定］カテゴリ

端子配置図の端子上にマウス・カーソルを移動した際，該当端子に関する情報をポップアップ表示させるか否かを選択します。

ツール・チップの表示設定	端子配置図の端子上にマウス・カーソルを移動した際，該当端子に関する情報をポップアップ表示させるか否かを選択します。	
	すべて表示	端子配置表の“説明”，“未使用時の処置方法”，“注意事項”に記載されている文字列を表示します。
	説明／未使用時の処置方法のみ	端子配置表の“説明”，“未使用時の処置方法”に記載されている文字列を表示します。
	注意事項のみ	端子配置表の“注意事項”に記載されている文字列を表示します。
	表示しない	端子上にマウス・カーソルを移動しても，何も表示しません。

(3)［端子名表示設定］カテゴリ

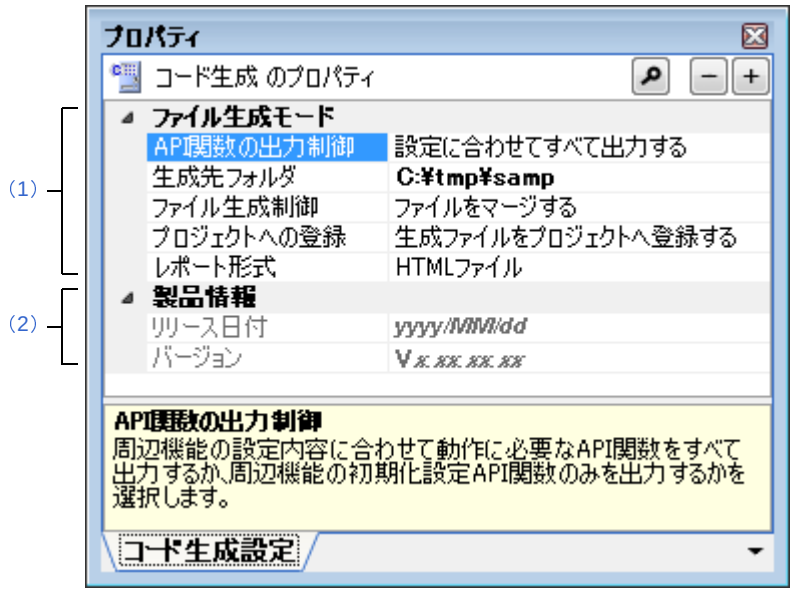
端子の付加情報を端子配置図に表示するか否かを選択します。

定義名	端子配置図の端子を，端子配置表の“定義名”に記載された文字列を付与した形式で表示するか否かを選択します。	
	表示する	端子配置表の“定義名”に記載されている文字列を付与した形式で表示します。
	表示しない	端子配置表の“定義名”に記載されている文字列を付与しません。
兼用機能	端子配置表の“選択機能”で機能を選択した際，非選択機能についても端子配置図に表示するか否かを選択します。	
	すべて	端子配置表の“選択機能”で選択された機能をかっこで括った形式で表示します。
	選択機能のみ	端子配置表の“選択機能”で選択された機能のみを端子配置図に表示します。

［コード生成設定］タブ

プロジェクト・ツリー パネルで選択した［コード生成（設計ツール）］に対応した情報（ファイル生成モード，製品情報）の表示，および設定の変更を行います。

図 A―7 ［コード生成設定］タブ



ここでは，次の項目について説明します。

- ［オープン方法］
- ［各エリアの説明］

［オープン方法］

- プロジェクト・ツリー パネルにおいて，［*Project name*（プロジェクト）］→［コード生成（設計ツール）］を選択したのち，［表示］メニュー→［プロパティ］を選択
- プロジェクト・ツリー パネルにおいて，［*Project name*（プロジェクト）］→［コード生成（設計ツール）］を選択したのち，コンテキスト・メニューから［プロパティ］を選択

備考　すでに本パネルがオープンしていた場合，プロジェクト・ツリー パネルの［コード生成（設計ツール）］を選択することにより，表示内容が切り替わります。

［各エリアの説明］

(1)［ファイル生成モード］カテゴリ

プロジェクト・ツリー パネルで選択した［コード生成（設計ツール）］に関する情報（API 関数の出力制御，生成先フォルダ，ファイル生成制御，プロジェクトへの登録，レポート形式）の表示，および設定の変更を行います。

API 関数の出力制御	出力する API 関数の種類を選択します。	
	設定に合わせてすべて出力する	周辺機能 パネルで使用する旨の設定が行われている周辺機能（クロック発生回路，ポート機能など）の全 API 関数を出力します。
	初期化関数のみ出力する	周辺機能 パネルで使用する旨の設定が行われている周辺機能（クロック発生回路，ポート機能など）の API 関数のうち，初期化処理に関する API 関数のみを出力します。
生成先フォルダ	ファイルの出力先を入力します。	
ファイル生成制御	周辺機能 パネルの コードを生成する ボタンがクリックされた際，既に同一のファイル名を有するファイルが存在していた場合の対応を選択します。	
	すでにファイルがあれば何もしない	同一のファイル名を有するファイルが存在していた場合，該当ファイルの出力を行いません。
	ファイルをマージする	同一のファイル名を有するファイルが存在していた場合，該当ファイルをマージします。 なお，マージする部位については， /* Start user code …. Do not edit comment generated here */ から /* End user code. Do not edit comment generated here */ で囲まれた部位に限られます。
	ファイルを上書きする	同一のファイル名を有するファイルが存在していた場合，該当ファイルを上書きします。
プロジェクトへの登録	周辺機能 パネルの コードを生成する ボタンがクリックされた際に出力されたファイルをプロジェクトに登録するか否かを選択します。	
	生成ファイルをプロジェクトへ登録する	登録します。
	生成ファイルをプロジェクトへ登録しない	登録しません。
レポート形式	［ファイル］メニュー→［コード生成レポート を保存］を選択した際に出力されるレポート・ファイル（2 種類：Function，Macro）の出力形式を選択します。	
	HTML ファイル	HTML 形式で出力します。
	CSV ファイル	CSV 形式で出力します。

備考　［ファイル生成制御］で［ファイルをマージする］を選択した場合，マージする部位内の“{”と“}”の数は一致させる必要があります。“{”と“}”の数が不一致な際は，正しいマージ処理が行われません。

(2)［製品情報］カテゴリ

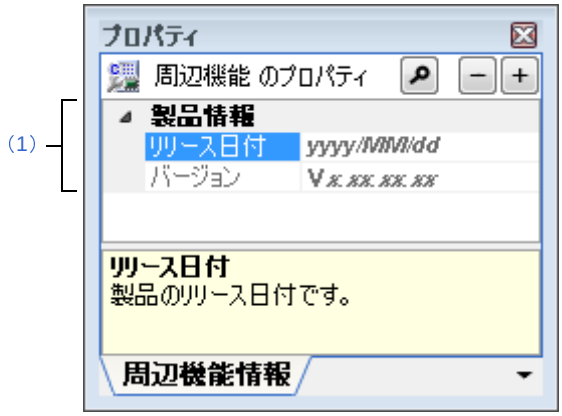
プロジェクト・ツリー パネルで選択した［コード生成（設計ツール）］に関する情報（リリース日付，バージョン）の表示を行います。

リリース日付	コード生成（設計ツール）のリリース日付を表示します。
バージョン	コード生成（設計ツール）のバージョンを表示します。

［周辺機能情報］タブ（製品情報）

プロジェクト・ツリー パネルで選択した［周辺機能］に対応した情報（製品情報）の表示を行います。

図 A—8　［周辺機能情報］タブ（製品情報）



ここでは，次の項目について説明します。

- ［オープン方法］
- ［各エリアの説明］

［オープン方法］

- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［コード生成（設計ツール）］→［周辺機能］を選択したのち，［表示］メニュー→［プロパティ］を選択
- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［コード生成（設計ツール）］→［周辺機能］を選択したのち，コンテキスト・メニューから［プロパティ］を選択

備考　すでに本パネルがオープンしていた場合，プロジェクト・ツリー パネルの［周辺機能］を選択することにより，表示内容が該当ノードに対応したものへと切り替わります。

［各エリアの説明］

(1)［製品情報］カテゴリ

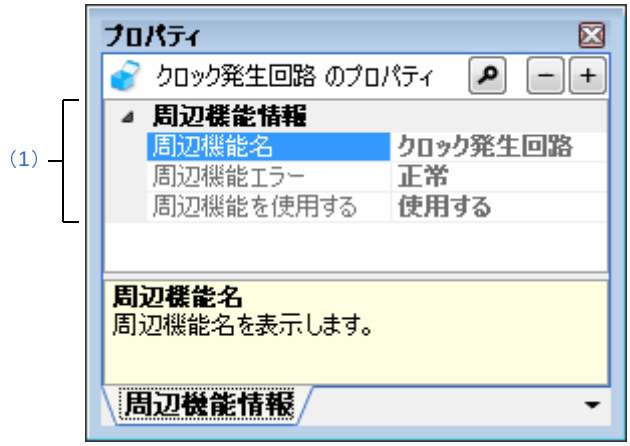
プロジェクト・ツリー パネルで選択した［周辺機能］に関する情報（リリース日付，バージョン）の表示を行います。

リリース日付	コード生成（設計ツール）のリリース日付を表示します。
バージョン	コード生成（設計ツール）のバージョンを表示します。

［周辺機能情報］タブ（周辺機能情報）

プロジェクト・ツリー パネルで選択した周辺機能ノードに対応した情報（周辺機能情報）の表示を行います。

図 A―9 ［周辺機能情報］タブ（周辺機能情報）



ここでは，次の項目について説明します。

- ［オープン方法］
- ［各エリアの説明］

［オープン方法］

- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［コード生成（設計ツール）］→［周辺機能］→周辺機能ノードを選択したのち，［表示］メニュー→［プロパティ］を選択
- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［コード生成（設計ツール）］→［周辺機能］→周辺機能ノードを選択したのち，コンテキスト・メニューから［プロパティ］を選択

備考　すでに本パネルがオープンしていた場合，プロジェクト・ツリー パネルの周辺機能ノードを選択することにより，表示内容が該当ノードに対応したものへと切り替わります。

［各エリアの説明］

(1)［周辺機能情報］カテゴリ

プロジェクト・ツリー パネルで選択した周辺機能ノードに関する情報（周辺機能名，周辺機能エラー，周辺機能を使用する）の表示を行います。

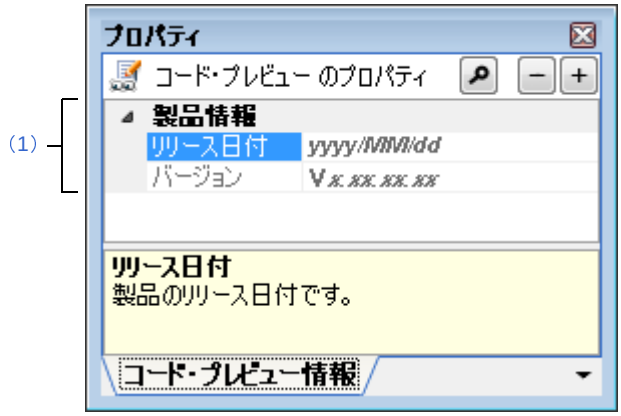
周辺機能名	周辺機能名を表示します。

周辺機能エラー	周辺機能 パネルの設定が正常に行われているか否かを表示します。	
	正常	不正な設定は検出されていません。
	入力エラー	不正な設定が検出されています。
周辺機能を使用する	周辺機能を使用するか否かを表示します。 なお，。使用の有無は，該当ノードに対応した周辺機能 パネルの設定内容に依存します。	
	使用する	周辺機能を使用します。
	いいえ	周辺機能を使用しません。

［コード・プレビュー情報］タブ（製品情報）

プロジェクト・ツリー パネルで選択した［コード・プレビュー］に対応した情報（製品情報）の表示を行います。

図 A―10　［コード・プレビュー情報］タブ（製品情報）



ここでは，次の項目について説明します。

- ［オープン方法］
- ［各エリアの説明］

［オープン方法］

- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［コード生成（設計ツール）］→［コード・プレビュー］を選択したのち，［表示］メニュー→［プロパティ］を選択
- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［コード生成（設計ツール）］→［コード・プレビュー］を選択したのち，コンテキスト・メニューから［プロパティ］を選択

備考　すでに本パネルがオープンしていた場合，プロジェクト・ツリー パネルの［コード・プレビュー］を選択することにより，表示内容が該当ノードに対応したものへと切り替わります。

［各エリアの説明］

(1)［製品情報］カテゴリ

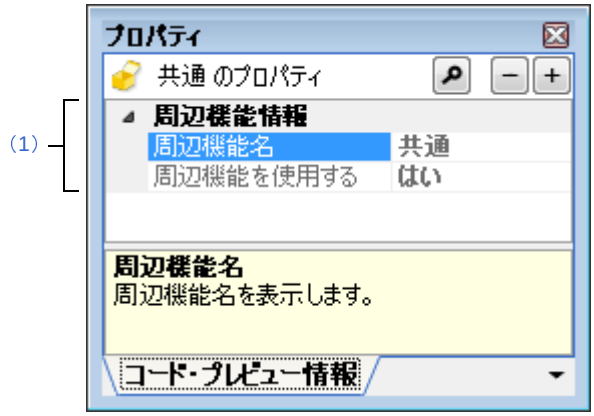
プロジェクト・ツリー パネルで選択した［コード・プレビュー］に関する情報（リリース日付，バージョン）の表示を行います。

リリース日付	コード生成（設計ツール）のリリース日付を表示します。
バージョン	コード生成（設計ツール）のバージョンを表示します。

［コード・プレビュー情報］タブ（周辺機能情報）

プロジェクト・ツリー パネルで選択した周辺機能ノードに対応した情報（周辺機能情報）の表示を行います。

図 A―11　［コード・プレビュー情報］タブ（周辺機能情報）



ここでは，次の項目について説明します。

- ［オープン方法］
- ［各エリアの説明］

［オープン方法］

- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［コード生成（設計ツール）］→［コード・プレビュー］→周辺機能ノードを選択したのち，［表示］メニュー→［プロパティ］を選択
- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［コード生成（設計ツール）］→［コード・プレビュー］→周辺機能ノードを選択したのち，コンテキスト・メニューから［プロパティ］を選択

備考　すでに本パネルがオープンしていた場合，プロジェクト・ツリー パネルの周辺機能ノードを選択することにより，表示内容が該当ノードに対応したものへと切り替わります。

［各エリアの説明］

(1)［周辺機能情報］カテゴリ

プロジェクト・ツリー パネルで選択した周辺機能ノードに関する情報（周辺機能名，周辺機能を使用する）の表示を行います。

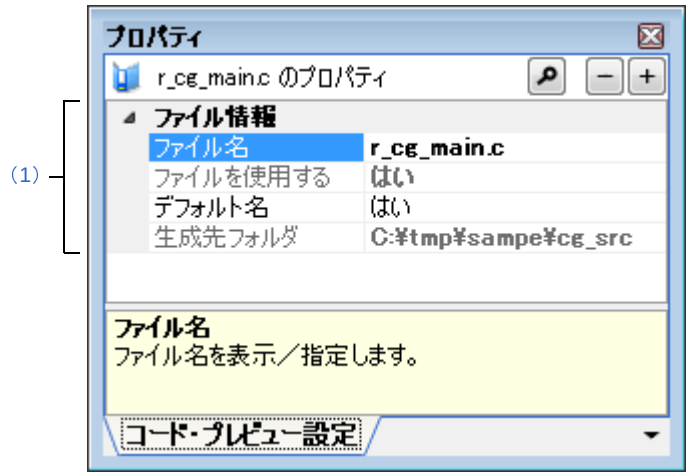
周辺機能名	周辺機能名を表示します。

周辺機能を使用する	周辺機能を使用するか否かを表示します。 なお，使用の有無は，該当ノードに対応した周辺機能 パネルの設定内容に依存します。	
	はい	周辺機能を使用します。
	いいえ	周辺機能を使用しません。

［コード・プレビュー設定］タブ（ファイル情報）

プロジェクト・ツリー パネルで選択したソース・コード・ノードに対応した情報（ファイル情報）の表示，および設定の変更を行います。

図 A―12　［コード・プレビュー設定］タブ（ファイル情報）



ここでは，次の項目について説明します。

- ［オープン方法］
- ［各エリアの説明］

［オープン方法］

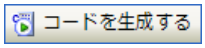
- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［コード生成（設計ツール）］→［コード・プレビュー］→周辺機能ノード→ソース・コード・ノードを選択したのち，［表示］メニュー→［プロパティ］を選択
- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［コード生成（設計ツール）］→［コード・プレビュー］→周辺機能ノード→ソース・コード・ノードを選択したのち，コンテキスト・メニューから［プロパティ］を選択

備考　すでに本パネルがオープンしていた場合，プロジェクト・ツリー パネルのソース・コード・ノードを選択することにより，表示内容が該当ノードに対応したものへと切り替わります。

［各エリアの説明］

(1)［ファイル情報］カテゴリ

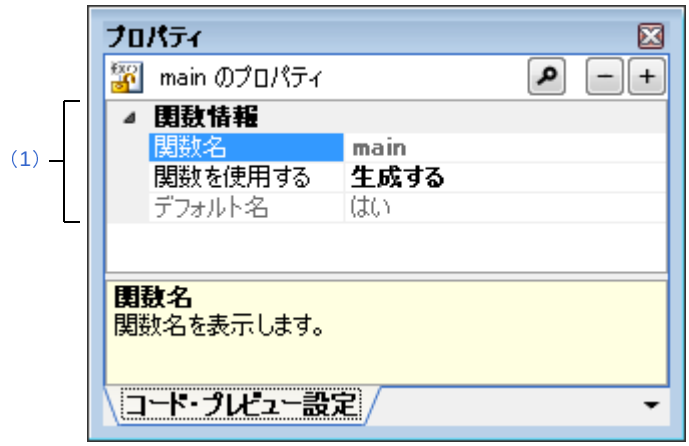
プロジェクト・ツリー パネルで選択したソース・コード・ノードに関する情報（ファイル名，ファイルを使用する，デフォルト名，生成先フォルダ）の表示，および設定の変更を行います。

<table>
<tr><td>ファイル名</td><td colspan="2">ファイル名を入力します。
なお，ファイル名は，プロジェクト・ツリー パネルでソース・コード・ノードを選択後，コンテキスト・メニューから［名前を変更する］を選択することにより変更することも可能です。</td></tr>
<tr><td rowspan="3">ファイルを使用する</td><td colspan="2">周辺機能 パネルの  ボタンがクリックされた際，ファイルを出力するか否かを表示します。
なお，使用の有無は，該当ノードに対応した周辺機能 パネルの設定内容に依存します。</td></tr>
<tr><td>はい</td><td>ファイルを出力します。</td></tr>
<tr><td>いいえ</td><td>ファイルを出力しません。</td></tr>
<tr><td rowspan="3">デフォルト名</td><td colspan="2">ファイル名をデフォルトの名前に戻すか否かを選択します。
なお，ファイル名は，プロジェクト・ツリー パネルでソース・コード・ノードを選択後，コンテキスト・メニューから［名前を元に戻す］を選択することによりデフォルトの名前に戻すことも可能です。</td></tr>
<tr><td>はい</td><td>デフォルトの名前に戻します。</td></tr>
<tr><td>いいえ</td><td>デフォルトの名前に戻しません。</td></tr>
<tr><td>生成先フォルダ</td><td colspan="2">ファイルの出力先を表示します。
なお，出力先は，［コード生成設定］タブ→［ファイル生成モード］→［生成先フォルダ］で変更することが可能です。</td></tr>
</table>

［コード・プレビュー設定］タブ（関数情報）

プロジェクト・ツリー パネルで選択した API 関数ノードに対応した情報（関数情報）の表示，および設定の変更を行います。

図 A―13　［コード・プレビュー設定］タブ（関数情報）



ここでは，次の項目について説明します。

- ［オープン方法］
- ［各エリアの説明］

［オープン方法］

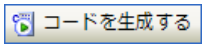
- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［コード生成（設計ツール）］→［コード・プレビュー］→周辺機能ノード→ソース・コード・ノード→ API 関数ノードを選択したのち，［表示］メニュー→［プロパティ］を選択
- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［コード生成（設計ツール）］→［コード・プレビュー］→周辺機能ノード→ソース・コード・ノード→ API 関数ノードを選択したのち，コンテキスト・メニューから［プロパティ］を選択

備考　すでに本パネルがオープンしていた場合，プロジェクト・ツリー パネルの API 関数ノードを選択することにより，表示内容が該当ノードに対応したものへと切り替わります。

［各エリアの説明］

（1）［関数情報］カテゴリ

プロジェクト・ツリー パネルで選択した API 関数ノードに関する情報（関数名，関数を使用する，デフォルト名）の表示，および設定の変更を行います。

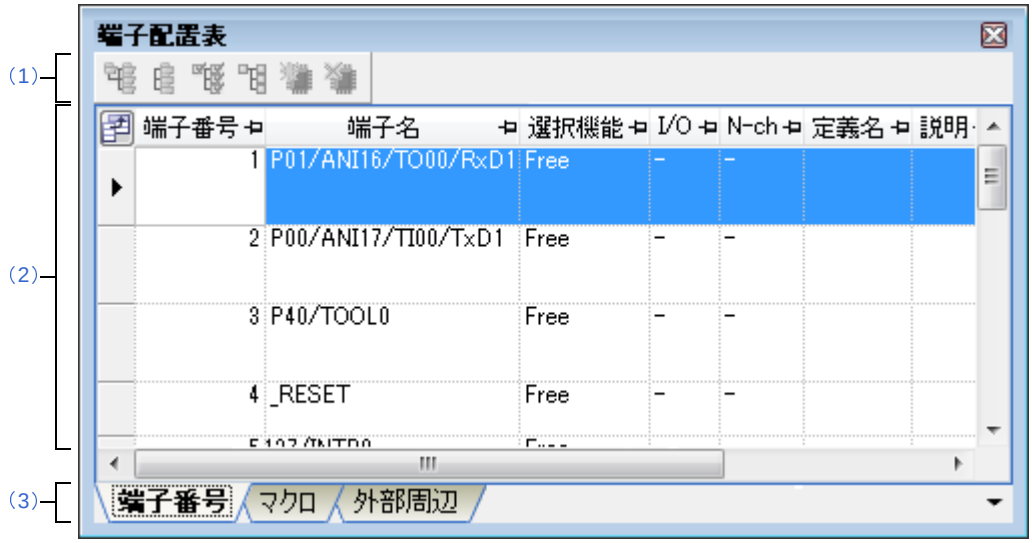
<table>
<tr><td>関数名</td><td colspan="2">API 関数名を入力します。
なお，API 関数名は，プロジェクト・ツリー パネルで API 関数ノードを選択後，コンテキスト・メニューから［名前を変更する］を選択することにより変更することも可能です。</td></tr>
<tr><td rowspan="3">関数を使用する</td><td colspan="2">周辺機能 パネルの  ボタンがクリックされた際，API 関数をファイルに出力するか否かを選択します。</td></tr>
<tr><td>生成する</td><td>API 関数を出力します。</td></tr>
<tr><td>生成しない</td><td>API 関数を出力しません。</td></tr>
<tr><td rowspan="3">デフォルト名</td><td colspan="2">API 関数名をデフォルトの名前に戻すか否かを選択します。
なお，API 関数名は，プロジェクト・ツリー パネルでソース・コード・ノードを選択後，コンテキスト・メニューから［名前を元に戻す］を選択することによりデフォルトの名前に戻すことも可能です。</td></tr>
<tr><td>はい</td><td>デフォルトの名前に戻します。</td></tr>
<tr><td>いいえ</td><td>デフォルトの名前に戻しません。</td></tr>
</table>

端子配置表 パネル

マイクロコントローラの各端子に関する情報を記述します。

備考　ツールバーの 100% ，または［Ctrl］キーを押下しながらマウス・ホイールを操作することにより，端子配置表エリアの内容を拡大／縮小することができます。

図 A―14　端子配置表 パネル



ここでは，次の項目について説明します。

- ［オープン方法］
- ［各エリアの説明］

［オープン方法］

- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［端子配置（設計ツール）］→［端子配置表］をダブルクリック
- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［端子配置（設計ツール）］→［端子配置表］を選択したのち，［Enter］キーを押下
- ［表示］メニュー→［端子配置］→［端子配置表］を選択

［各エリアの説明］

（1）ツールバー

本エリアは，以下に示したボタン群から構成されています。

	端子配置表エリアの情報を展開表示します。

	端子配置表エリアの情報を折りたたみ表示します。
	［マクロ］タブの第１階層に表示されている周辺機能を選択したのち，本ボタンをクリックすることにより，選択機能，I/O，N-ch などといった各欄に対する情報の設定処理が自動実行されます。
	［マクロ］タブの第１階層に表示されている周辺機能を選択したのち，本ボタンをクリックすることにより，選択機能，I/O，N-ch などといった各欄の情報が初期化されます。
	本ボタンをクリックすることにより，［外部周辺］タブに外部周辺コントローラに関する情報が，端子配置図 パネルに外部周辺コントローラが作成表示されます。
	［外部周辺］タブの第１階層に表示されている外部周辺コントローラを選択したのち，本ボタンをクリックすることにより，該当情報が削除されます。

備考 1.  ボタンをクリックした際には，［端子番号］タブ，および［マクロ］タブの“外部周辺”列の選択肢として該当情報が追加されます。

2.  ボタンをクリックした際には，端子配置図 パネルの端子配置図エリアから該当外部周辺部品が削除されます。

(2) 端子配置表エリア

マイクロコントローラの各端子に関する情報を記述するための“端子配置表”を表示します。

(3) タブ選択エリア

タブを選択することにより，“マイクロコントローラの各端子に関する情報”の表示順序が切り替わります。本パネルには，次のタブが存在します。

- ［端子番号］タブ

  マイクロコントローラの各端子に関する情報を端子番号順で表示

- ［マクロ］タブ

  マイクロコントローラの各端子に関する情報を周辺機能単位にグルーピングされた順序で表示

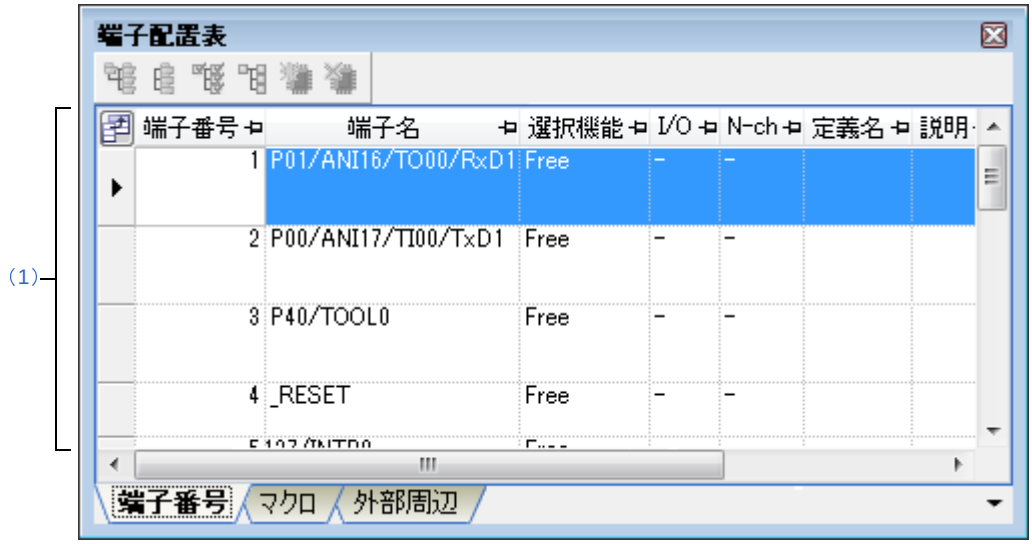
- ［外部周辺］タブ

  外部周辺に接続された端子に関する情報を外部周辺部品単位にグルーピングされた順序で表示

［端子番号］タブ

マイクロコントローラの各端子に関する情報を端子番号順で表示します。

図 A―15 ［端子番号］タブ



ここでは，次の項目について説明します。

- ［オープン方法］
- ［各エリアの説明］

［オープン方法］

- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［端子配置（設計ツール）］→［端子配置表］をダブルクリック
- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［端子配置（設計ツール）］→［端子配置表］を選択したのち，［Enter］キーを押下
- ［表示］メニュー→［端子配置］→［端子配置表］を選択

［各エリアの説明］

（1）端子配置表エリア

マイクロコントローラの各端子に関する情報を記述するための“端子配置表”を表示します。

なお，本エリアの端子配置表は，端子番号順となっています。

以下に，端子配置表を構成する列を示します。

列の見出し	概要
端子番号	該当端子の端子番号を表示します。

列の見出し	概要
端子名	該当端子の端子名を表示します。
選択機能	該当端子が複数の機能を有している際，“どのような機能で利用するのか”を選択するための領域です。
I/O	該当端子の入出力モードを選択するための領域です。
N-ch	該当端子を出力モードで使用する際，“どのような出力モードで利用するのか”を選択するための領域です。
定義名	該当端子に“ユーザ独自の端子名”を付与するための領域です。 なお，定義名として入力可能な文字数は，256 文字までに限られます。
説明	該当端子の機能概要を表示します。
未使用時の処置方法	該当端子を使用しない場合の処置方法を表示します。 なお，本欄は，“選択機能”欄で Free が選択されている場合に限り表示されます。
注意事項	該当端子を使用するうえで注意すべき事項を表示します。
外部周辺	該当端子を“どの外部周辺コントローラに接続するのか”を選択するための領域です。

備考 1. “端子番号”，“端子名”，“説明”，“未使用時の処置方法”，“注意事項”については，固定化された情報のため，該当欄に情報を追記することはできません。

2. “選択機能”欄の Free を固有端子名に変更した場合，端子配置図 パネルの該当端子色がプロパティ パネルの［端子配置図の設定］タブ→［色設定］で選択された“未使用端子の表示色”から“使用端子の表示”へと変化します。

3. 列の移動（表示順序の変更）は，端子配置表の該当列をドラッグしたのち，移動先にドロップすることにより行います。

4. “ユーザ独自の列”を追加する場合，端子配置表の左上に設けられた  ボタンをクリックすることによりオープンする列の選択 ダイアログの［新しい列 ...］ボタンをクリックすることによりオープンする新しい列 ダイアログで行います。

［マクロ］タブ

マイクロコントローラの各端子に関する情報を周辺機能単位にグルーピングされた順序で表示します。

図 A―16　［マクロ］タブ

ここでは，次の項目について説明します。

- ［オープン方法］
- ［各エリアの説明］

［オープン方法］

- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［端子配置（設計ツール）］→［端子配置表］をダブルクリック
- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［端子配置（設計ツール）］→［端子配置表］を選択したのち，［Enter］キーを押下
- ［表示］メニュー→［端子配置］→［端子配置表］を選択

［各エリアの説明］

（1）端子配置表エリア

マイクロコントローラの各端子に関する情報を記述するための“端子配置表”を表示します。

なお，本エリアの端子配置表は，周辺機能単位にグルーピングされた順序となっています。

（a）第 1 階層

以下に，端子配置表を構成する列を示します。

列の見出し	概要
マクロ名	周辺機能の名称を表示します。
総数	周辺機能に対して割り当てられている端子の総数を表示します。
使用中	用途が設定済みの端子の総数を表示します。
他で使用中	他の周辺機能で用途が設定済みの端子の総数を表示します。

（b）第 2 階層

列の見出し	概要
端子番号	該当端子の端子番号を表示します。
端子名	該当端子の端子名を表示します。
選択機能	該当端子が複数の機能を有している際，“どのような機能で利用するのか”を選択するための領域です。
I/O	該当端子の入出力モードを選択するための領域です。
N-ch	該当端子を出力モードで使用する際，“どのような出力モードで利用するのか”を選択するための領域です。
定義名	該当端子に“ユーザ独自の端子名”を付与するための領域です。 なお，定義名として入力可能な文字数は，256 文字までに限られます。
説明	該当端子の機能概要を表示します。
未使用時の処置方法	該当端子を使用しない場合の処置方法を表示します。 なお，本欄は，“選択機能”欄で Free が選択されている場合に限り表示されます。
注意事項	該当端子を使用するうえで注意すべき事項を表示します。
外部周辺	該当端子を“どの外部周辺コントローラに接続するのか”を選択するための領域です。

備考 1. “マクロ名”，“総数”，“使用中”，“他で使用中”，“端子番号”，“端子名”，“説明”，“未使用時の処置方法”，“注意事項”については，固定化された情報のため，該当欄に情報を追記することはできません。

2. “選択機能”欄の Free を固有端子名に変更した場合，端子配置図 パネルの該当端子色がプロパティ パネルの［端子配置図の設定］タブ→［色設定］で選択された“未使用端子の表示色”から“使用端子の表示”へと変化します。

3. 列の移動（表示順序の変更）は，端子配置表の該当列をドラッグしたのち，移動先にドロップすることにより行います。

4. “ユーザ独自の列”を追加する場合，端子配置表の左上に設けられた  ボタンをクリックすることによりオープンする列の選択 ダイアログの［新しい列 ...］ボタンをクリックすることによりオープンする新しい列 ダイアログで行います。

［外部周辺］タブ

外部周辺に接続された端子に関する情報を外部周辺部品単位にグルーピングされた順序で表示します。

図 A―17　［外部周辺］タブ

ここでは，次の項目について説明します。

- ［オープン方法］
- ［各エリアの説明］

［オープン方法］

- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［端子配置（設計ツール）］→［端子配置表］をダブルクリック
- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［端子配置（設計ツール）］→［端子配置表］を選択したのち，［Enter］キーを押下
- ［表示］メニュー→［端子配置］→［端子配置表］を選択

［各エリアの説明］

（1）端子配置表エリア

外部周辺コントローラの各端子に関する情報を記述するための“端子配置表”を表示します。

なお，本エリアの端子配置表は，外部周辺コントローラ単位にグルーピングされた順序となっています。

（a）第 1 階層

以下に，端子配置表を構成する列を示します。

列の見出し	概要
外部周辺名	外部周辺コントローラの名称を表示します。 なお，名称を変更する場合は，本欄を選択したのち，［F2］キーを押下することにより行います。
総数	マイクロコントローラとの接続用に割り当てられている端子の総数を表示します。

（b）第 2 階層

列の見出し	概要
端子番号	該当端子の端子番号を表示します。
端子名	該当端子の端子名を表示します。
選択機能	該当端子が複数の機能を有している際，“どのような機能で利用するのか”を選択するための領域です。
I/O	該当端子の入出力モードを選択するための領域です。
N-ch	該当端子を出力モードで使用する際，“どのような出力モードで利用するのか”を選択するための領域です。
定義名	該当端子に“ユーザ独自の端子名”を付与するための領域です。 なお，定義名として入力可能な文字数は，256 文字までに限られます。
説明	該当端子の機能概要を表示します。
未使用時の処置方法	該当端子を使用しない場合の処置方法を表示します。 なお，本欄は，“選択機能”欄で Free が選択されている場合に限り表示されます。
注意事項	該当端子を使用するうえで注意すべき事項を表示します。

備考 1. “接続数”，“端子番号”，“端子名”，“説明”，“未使用時の処置方法”，“注意事項”については，固定化された情報のため，該当欄に情報を追記することはできません。

2. “選択機能”欄の Free を固有端子名に変更した場合，端子配置図 パネルの該当端子色がプロパティ パネルの［端子配置図の設定］タブ→［色設定］で選択された“未使用端子の表示色”から“使用端子の表示”へと変化します。

3. 列の移動（表示順序の変更）は，端子配置表の該当列をドラッグしたのち，移動先にドロップすることにより行います。

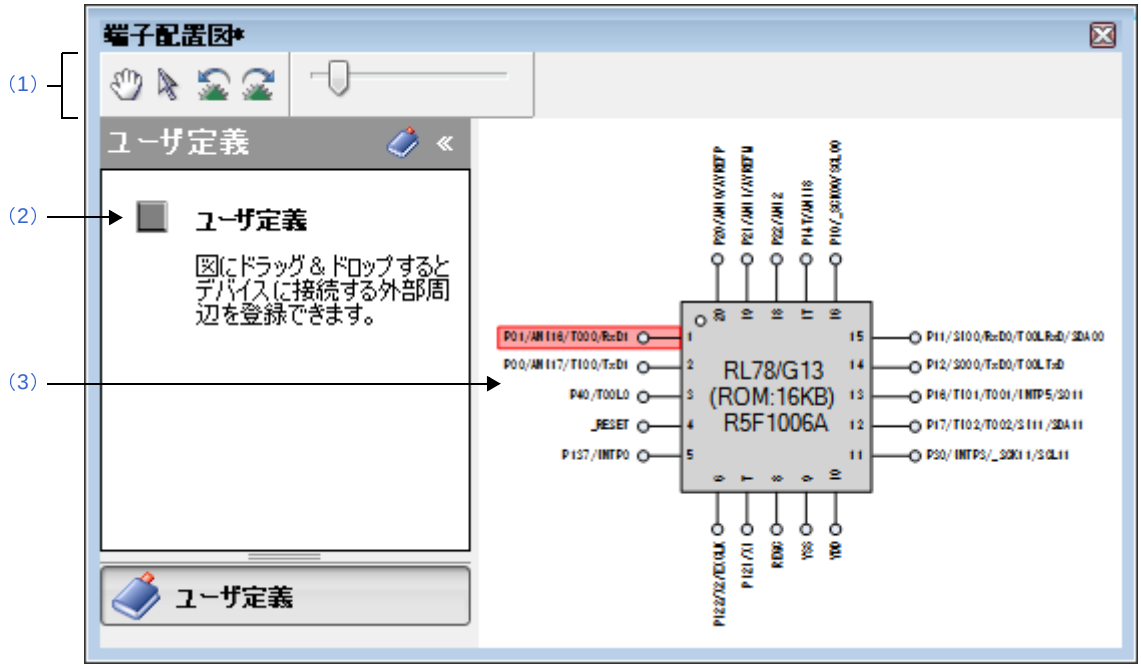
4. “ユーザ独自の列”を追加する場合，端子配置表の左上に設けられた  ボタンをクリックすることによりオープンする列の選択 ダイアログの［新しい列 ...］ボタンをクリックすることによりオープンする新しい列 ダイアログで行います。

端子配置図 パネル

端子配置表 パネルにおける情報の記述状況を表示します。

備考　ツールバーの 100% により，端子配置図エリアの内容を拡大／縮小することができます。

図 A—18　端子配置図 パネル



ここでは，次の項目について説明します。

- ［オープン方法］
- ［各エリアの説明］
- ［コンテキスト・メニュー］

［オープン方法］

- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［端子配置（設計ツール）］→［端子配置図］をダブルクリック
- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［端子配置（設計ツール）］→［端子配置図］を選択したのち，［Enter］キーを押下
- ［表示］メニュー→［端子配置］→［端子配置図］を選択

備考　プロパティ パネルの［端子配置の設定］タブ→［パッケージ情報］→［パッケージ名］に“BGA”を選択している場合，本パネルをオープンすることができません。

[各エリアの説明]

(1) ツールバー

本エリアは，以下に示したボタン群から構成されています。

	本ボタンをクリックすることにより，ドラッグ・アンド・ドロップで端子配置図エリアの表示部分を変更することが可能となります。 なお，本ボタンのクリックにより，端子配置図エリア内におけるマウス・カーソルの形状が矢印から手形へと変化します。
	本ボタンをクリックすることにより，端子配置図エリアに表示されている外部周辺部品を任意の位置に移動したり，端子を選択したりすることが可能となります。 なお，本ボタンのクリックにより，  ボタンのクリックにより変化したマウス・カーソルの形状が手形から矢印へと戻ります。
	端子配置図エリアの内容を左に 90 度回転します。
	端子配置図エリアの内容を右に 90 度回転します。
	端子配置図エリアの内容を拡大／縮小します。

(2) [ユーザ定義] エリア

本エリア内のボタンを端子配置図エリアにドラッグ・アンド・ドロップすることにより，外部周辺コントローラが作成表示されます。

(3) 端子配置図エリア

マイクロコントローラの端子配置状況を表示します。

なお，端子配置の設定状況については，プロパティ パネルの［端子配置図の設定］タブ→［色設定］で指定された色での表示となります。

備考　図中の端子名をダブルクリックした際には，端子配置表 パネルがオープンし，表中の該当端子にフォーカスが遷移します。

[コンテキスト・メニュー]

端子配置図エリアの端子上，または外部周辺コントローラ上でマウスを右クリックすることにより表示されるコンテキスト・メニューは，以下のとおりです。

(1) 端子上で右クリックした場合

機能選択	該当端子が複数の機能を有している際，“どのような機能で利用するのか”を選択します。
外部周辺選択	該当端子を“どの外部周辺コントローラに接続するのか”を選択します。

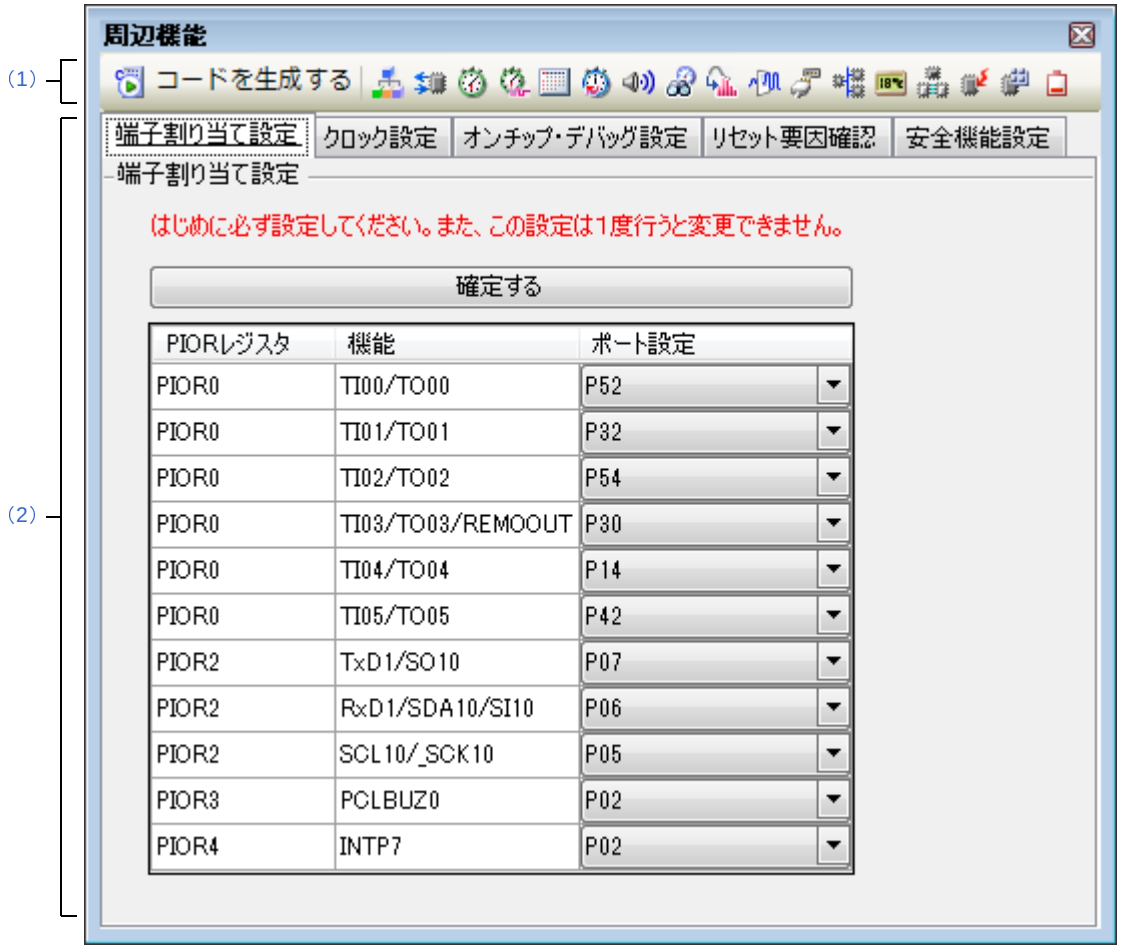
（2）外部周辺コントローラ上で右クリックした場合

接続端子の切断	該当端子との接続を切断します。
外部周辺削除	外部周辺コントローラを削除します。

周辺機能 パネル

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情報を設定します。

図 A—19　周辺機能 パネル



ここでは，次の項目について説明します。

- ［オープン方法］
- ［各エリアの説明］

［オープン方法］

- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［コード生成（設計ツール）］→［周辺機能］（→周辺機能ノード）をダブルクリック
- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［コード生成（設計ツール）］→［周辺機能］（→周辺機能ノード）を選択したのち，［Enter］キーを押下
- ［表示］メニュー→［コード生成 2］→［周辺機能］を選択

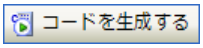
備考　すでに本パネルがオープンしていた場合，周辺機能ボタン“  ，  など”をクリックすることにより，情報設定エリアの表示内容が該当ボタンに対応したものへと切り替わります。

[各エリアの説明]

(1) ツールバー

本エリアは，以下に示したボタン群“周辺機能ボタン”から構成されています。

なお，対象マイクロコントローラが未サポートの周辺機能については，該当周辺機能ボタンが表示されません。

	プロパティ パネルの［コード生成設定］タブ→［ファイル生成モード］→［生成先フォルダ］で指定されたフォルダにソース・コード（デバイス・ドライバ・プログラム）を出力します。
 など	情報設定エリアの表示内容を該当周辺機能を制御するうえで必要な情報へと切り替えます。

(2) 情報設定エリア

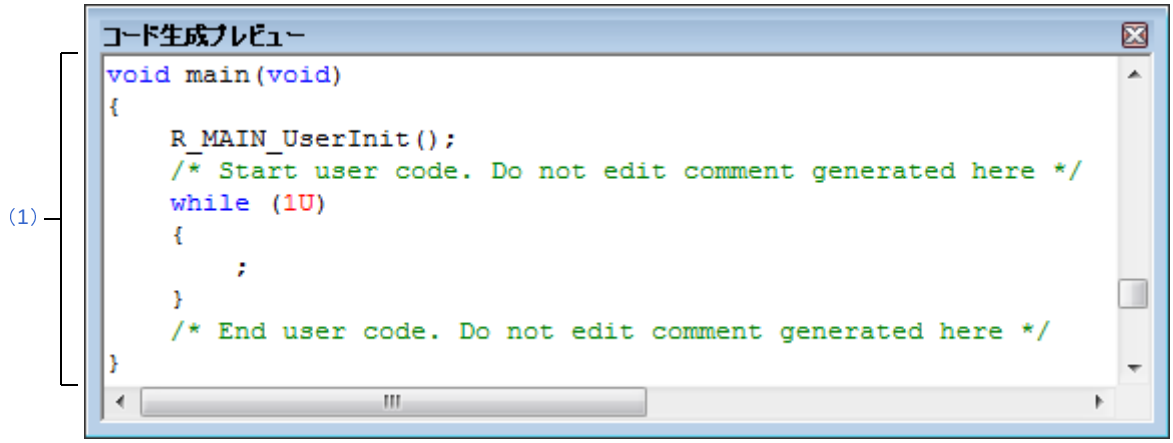
本エリアの表示内容については，本パネルをオープンする際に選択／クリックする“周辺機能ノード”，または“周辺機能ボタン”の種類により異なります。

なお，設定項目についての詳細は，マイクロコントローラのユーザーズ・マニュアルを参照してください。

コード生成プレビュー パネル

周辺機能 パネルの設定内容に応じたソース・コードを確認します。

図 A―20　コード生成プレビュー パネル



ここでは，次の項目について説明します。

- ［オープン方法］
- ［各エリアの説明］
- ［コンテキスト・メニュー］

［オープン方法］

- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［コード生成（設計ツール）］→［コード・プレビュー］→周辺機能ノード→ソース・コード・ノード（→ API 関数ノード）をダブルクリック
- プロジェクト・ツリー パネルの［*Project name*（プロジェクト）］→［コード生成（設計ツール）］→［コード・プレビュー］→周辺機能ノード→ソース・コード・ノード（→ API 関数ノード）を選択したのち，［Enter］キーを押下
- ［表示］メニュー→［コード生成 2］→［コード生成プレビュー］を選択

備考　すでに本パネルがオープンしていた場合，ソース・コード・ノード（→ API 関数ノード）をダブルクリックすることにより，ソース・コード表示エリアの表示内容が該当ノードに対応したものへと切り替わります。

［各エリアの説明］

（1）ソース・コード表示エリア

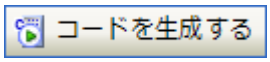
周辺機能 パネルで設定した情報に応じたソース・コード（デバイス・ドライバ・プログラム）の確認を行います。

なお，本エリアに表示されるソース・コードの文字色は，以下の意味を持ちます。

表 A―2　ソース・コードの文字色

文字色	概要
緑	コメント文
青	C コンパイラの予約語
赤	数値
黒	コード部
グレー	ファイル名

備考 1.　本パネル内でソース・コードを編集することはできません。

2.　一部の API 関数については，ソース・コードの出力時（周辺機能 パネルの 

ボタンをクリックした際）にレジスタ値などが計算され確定するものがあります。このため，本パネルに表示されるソース・コードは，実際に出力されるソース・コードと一致しない場合があります。

[コンテキスト・メニュー]

マウスを右クリックすることにより表示されるコンテキスト・メニューは，以下のとおりです。

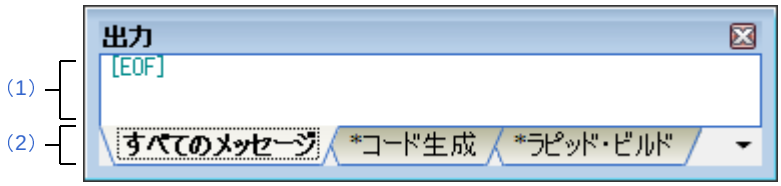
コピー	選択している文字列をクリップ・ボードに保存します。
すべて選択	ソース・コード表示エリアに表示されている全文字列を選択します。

出力 パネル

CubeSuite+ が提供している各種コンポーネント（設計ツール，ビルド・ツールなど）の操作ログを表示します。
メッセージは，出力元のツールごとに分類されたタブ上でそれぞれ個別に表示されます。

備考　ツールバーの 100% ，または［Ctrl］キーを押下しながらマウス・ホイールを操作することにより，メッセージ・エリアの内容を拡大／縮小することができます。

図 A―21　出力 パネル



ここでは，次の項目について説明します。

- ［オープン方法］
- ［各エリアの説明］
- ［コンテキスト・メニュー］

［オープン方法］

- ［表示］メニュー→［出力］を選択

［各エリアの説明］

(1) メッセージ・エリア

各ツールから出力されたメッセージを表示します。

なお，メッセージの文字色／背景色は，出力メッセージの種別により異なります（文字色／背景色はオプション ダイアログにおける［全般 - フォントと色］カテゴリの設定に依存）。

(2) タブ選択エリア

メッセージの出力元を示すタブを選択します。

設計ツールでは，次のタブを使用します。

タブ名	説明
すべてのメッセージ	CubeSuite+ が提供している全コンポーネント（設計ツール，ビルド・ツールなど）から出力されるメッセージを表示します（ラピッド・ビルドの実行によるメッセージを除く）。
コード生成	CubeSuite+ が提供している各種コンポーネント（設計ツールを含む，ビルド・ツール／デバッグ・ツール／解析ツールなど）から出力されるメッセージのうち，コード生成が出力するメッセージを表示します。

注意　新たなメッセージが非選択状態のタブ上に出力されても，自動的なタブの表示切り替えは行いません。この場合，タブ名の先頭に“*”が付加し，新たなメッセージが出力されていることを示します。

[コンテキスト・メニュー]

マウスを右クリックすることにより表示されるコンテキスト・メニューは，以下のとおりです。

コピー	選択している文字列をクリップ・ボードに保存します。
すべて選択	メッセージ・エリアに表示されている全文字列を選択します。
クリア	メッセージ・エリアに表示されている全文字列を消去します。
タグ・ジャンプ	キャレット行のメッセージに対応するエディタ（ファイル，行，桁）へジャンプします。
メッセージに関するヘルプ	メッセージに対応したヘルプを表示します。 ただし，本項目の選択は，キャレットが警告メッセージ／エラー・メッセージの表示行にある場合に限られます。

列の選択 ダイアログ

本ダイアログに表示されている項目を端子配置表に表示するか否かの選択，および端子配置表に対する列の翻訳追加／削除を行います。

図 A―22　列の選択 ダイアログ

ここでは，次の項目について説明します。

- ［オープン方法］
- ［各エリアの説明］
- ［機能ボタン］

［オープン方法］

- 端子配置表 パネルの［端子番号］タブにおいて，ボタンをクリック
- 端子配置表 パネルの［マクロ］タブにおいて，ボタンをクリック
- 端子配置表 パネルの［外部周辺］タブにおいて，ボタンをクリック

［各エリアの説明］

(1) 操作対象選択エリア

本ダイアログの操作対象となる端子配置表を選択します。

端子番号	［端子番号］タブの端子配置表を操作対象とします。
マクロ	［マクロ］タブの第 1 階層の端子配置表を操作対象とします。
マクロ - 端子	［マクロ］タブの第 2 階層の端子配置表を操作対象とします。

外部周辺	［外部周辺］タブの第 1 階層の端子配置表を操作対象とします。
外部周辺 - 端子	［外部周辺］タブの第 2 階層の端子配置表を操作対象とします。

（2）表示項目選択エリア

該当項目を操作対象選択エリアで選択された端子配置表に表示するか否かを選択します。

チェック状態	該当項目を端子配置表に表示します。
非チェック状態	該当項目を端子配置表から非表示とします。

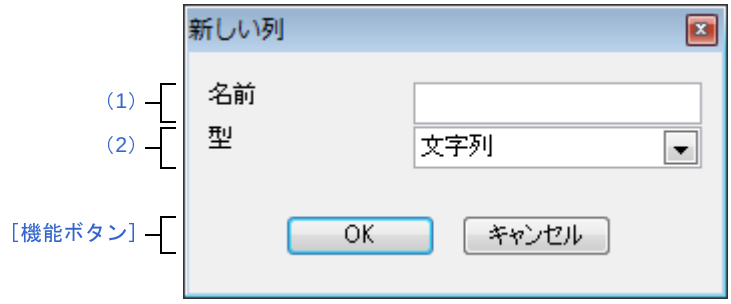
［機能ボタン］

ボタン	機能
新しい列 ...	端子配置表に列を追加するための新しい列 ダイアログをオープンします。
列の削除	選択された列を端子配置表から削除します。 なお，削除可能な列は，新しい列 ダイアログでユーザが独自に追加した列に限られます。
デフォルト	列の並び順を初期状態に戻します。
閉じる	本ダイアログをクローズします。

新しい列 ダイアログ

端子配置表に列を追加します。

図 A―23　新しい列 ダイアログ



ここでは，次の項目について説明します。

- ［オープン方法］
- ［各エリアの説明］
- ［機能ボタン］

［オープン方法］

- 列の選択 ダイアログの［新しい列 ...］ボタンをクリック

［各エリアの説明］

(1)［名前］

端子配置表に追加する列の見出しを入力します。

なお，名前として入力可能な文字数は，256 文字までに限られます。

(2)［型］

端子配置表に追加する列の入力フォームを選択します。

文字列	文字列のみ入力可能な列となります。
チェック・ボックス	チェック・ボックスの設けられた列となります。
整数	整数のみ入力可能な列となります。
実数	実数のみ入力可能な列となります。
日付	年月日形式の日付のみ入力可能な列となります。

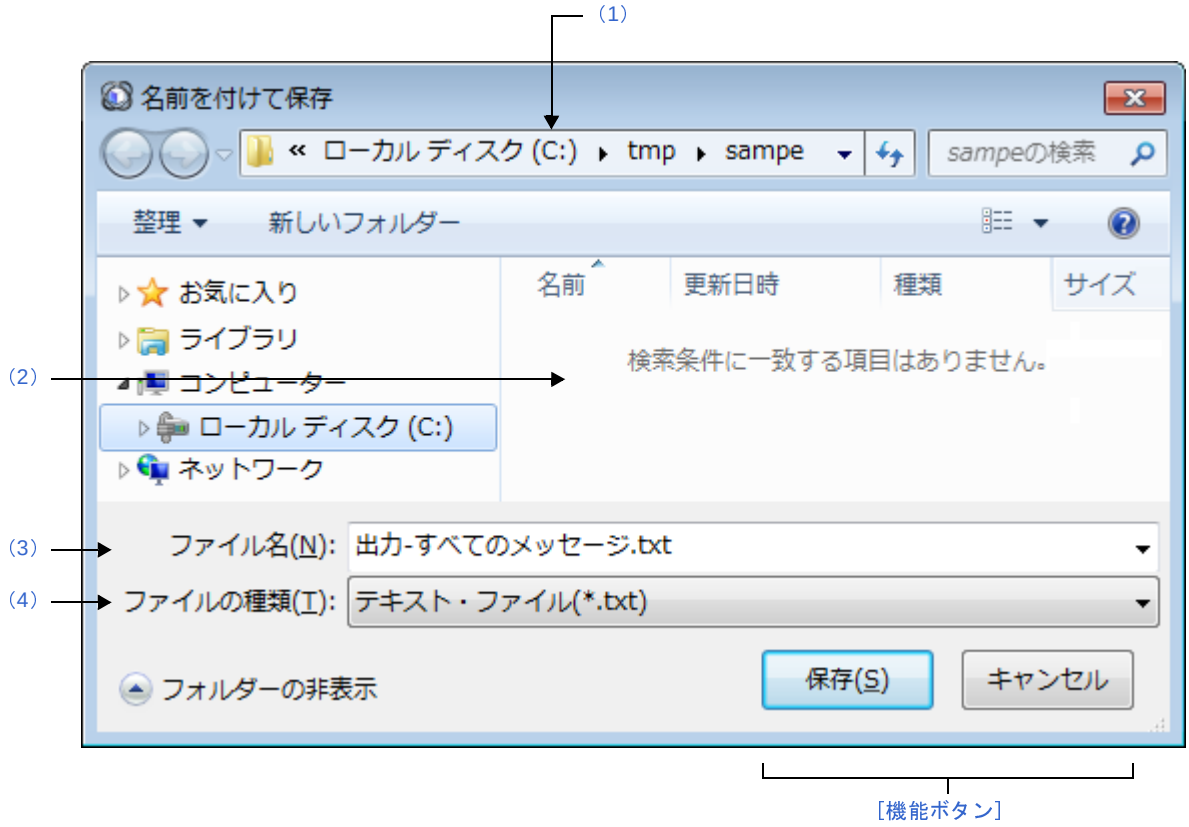
［機能ボタン］

ボタン	機能
OK	［名前］で指定された見出しを有する列を端子配置表の右端に追加します。
キャンセル	本ダイアログをクローズします。

名前を付けて保存 ダイアログ

ファイルに名前を付けて保存します。

図 A―24　名前を付けて保存 ダイアログ



ここでは，次の項目について説明します。

- ［オープン方法］
- ［各エリアの説明］
- ［機能ボタン］

［オープン方法］

- ［ファイル］メニュー→［出力 - *タブ名* を保存］を選択
- ［ファイル］メニュー→［名前を付けて 出力 - *タブ名* を保存 ...］を選択

［各エリアの説明］

(1) 保存する場所エリア

ファイルの保存先（フォルダ名）を選択します。

(2) ファイルの一覧エリア

保存する場所エリア，および［ファイルの種類］で選択された条件に合致するファイルの一覧を表示します。

(3)［ファイル名］

ファイルの名前（ファイル名）を指定します。

(4)［ファイルの種類］

ファイルの種類（ファイル・タイプ）を選択します。

［機能ボタン］

ボタン	機能
保存	保存する場所エリアで指定されたフォルダに［ファイル名］，および［ファイルの種類］で指定された名前のファイルを出力します。
キャンセル	本ダイアログをクローズします。

付録 B　出力ファイル

本付録では，コード生成が出力するファイルについて説明します。

B. 1　説　　明

以下に，コード生成が出力するファイルの一覧を示します。

表 B—1　出力ファイル

周辺機能	ファイル名	API 関数名
共　　通	r_main.c，または r_cg_main.c	main R_MAIN_UserInit
	r_systeminit.c，または r_cg_systeminit.c	hdwinit R_Systeminit
	r_cg_macrodriver.h	—
	r_cg_userdefine.h	—
	r_cg_lk.dr	—
クロック発生回路	r_cg_cgc.c	R_CGC_Create R_CGC_Set_ClockMode R_CGC_Set_CRCOn 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	—

周辺機能	ファイル名	API 関数名
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_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	R_TMR_RJ0_Create R_TMR_RJ0_Start R_TMR_RJ0_Stop R_TMR_RJ0_Set_PowerOff R_TMR_RJ0_Get_PulseWidth
	r_cg_timer_user.c	R_TMR_RJ0_Create_UserInit r_tmr_rj0_interrupt
	r_cg_timer.h	—
タイマ RD	r_cg_timer.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_cg_timer_user.c	R_TMR_RDn_Create_UserInit r_tmr_rdn_interrupt
	r_cg_timer.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	—

周辺機能	ファイル名	API 関数名
16 ビット・タイマ KB	r_cg_timer.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_cg_timer_user.c	R_TMR_KBm_Create_UserInit r_tmr_kbm_interrupt
	r_cg_timer.h	—
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_Set_BatchOverwriteRequestOn
	r_cg_kb2_user.c	R_KB2m_Create_UserInit r_kb2m_interrupt
	r_cg_kb2.h	—

周辺機能	ファイル名	API 関数名
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_Get_CounterValue R_RTC_Set_ConstPeriodInterruptOn R_RTC_Set_ConstPeriodInterruptOff R_RTC_Set_AlarmOn R_RTC_Set_AlarmOff R_RTC_Set_AlarmValue R_RTC_Get_AlarmValue R_RTC_Set_RTC1HZOn R_RTC_Set_RTC1HZOff
	r_cg_rtc_user.c	R_RTC_Create_UserInit r_rtc_interrupt r_rtc_callback_constperiod r_rtc_callback_alarm
	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	—
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_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_cg_it8bit_user.c	R_IT8bitm_Channeln_Create_UserInit r_it8bitm_channeln_interrupt
	r_cg_it8bit.h	—

周辺機能	ファイル名	API 関数名
16 ビット・ウエイクアップ・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_PCLBUZ_Set_PowerOff
	r_cg_pclbuz_user.c	R_PCLBUZn_Create_UserInit
	r_cg_pclbuz.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	—
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_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	—
温度センサ	r_cg_tmps.c	R_TMPS_Create R_TMPS_Start R_TMPS_Stop R_TMPS_Set_PowerOff
	r_cg_tmps_user.c	R_TMPS_Create_UserInit
	r_cg_tmps.h	—

周辺機能	ファイル名	API 関数名
24 ビット ΔΣA/D コンバータ	r_cg_dsadc.c	R_DSADC_Create R_DSADC_Set_OperationOn R_DSADC_Set_OperationOff R_DSADC_Start R_DSADC_Stop 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_cg_dsadc.h	—
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_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_cg_pga_user.c	R_PGA_Create_UserInit
	r_cg_pga.h	—
コンパレータ	r_cg_comp.c	R_COMP_Create R_COMPn_Start R_COMPn_Stop R_COMP_Set_PowerOff
	r_cg_comp_user.c	R_COMP_Create_UserInit r_compn_interrupt
	r_cg_comp.h	—

周辺機能	ファイル名	API 関数名
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_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	—

周辺機能	ファイル名	API 関数名
シリアル・アレイ・ユニット 4（DALI/UART4）	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	—
アシンクロナス・シリアル・インタフェース LIN-UART（UARTF）	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	—

周辺機能	ファイル名	API 関数名
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_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_serial_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_cg_lcd_user.c	R_LCD_Create_UserInit r_lcd_interrupt
	r_cg_lcd.h	—
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	—

周辺機能	ファイル名	API 関数名
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	—
DTC	r_cg_dtc.c	R_DTC_Create R_DTCn_Start R_DTCn_Stop R_DTC_Set_PowerOff
	r_cg_dtc_user.c	R_DTC_Create_UserInit
	r_cg_dtc.h	—
イベントリンクコントローラ（ELC）	r_cg_elc.c	R_ELC_Create R_ELC_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_cg_intc_user.c	R_INTC_Create_UserInit r_intcn_interrupt r_intclrn_interrupt
	r_cg_intc.h	—
キー割り込み機能	r_cg_intc.c	R_KEY_Create R_KEY_Start R_KEY_Stop
	r_cg_intc_user.c	R_KEY_Create_UserInit r_key_interrupt
	r_cg_intc.h	—
電圧検出回路	r_cg_lvd.c	R_LVD_Create R_LVD_InterruptMode_Start
	r_cg_lvd_user.c	R_LVD_Create_UserInit r_lvd_interrupt
	r_cg_lvd.h	—

周辺機能	ファイル名	API 関数名
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_bup_user.c	R_BUP_Create_UserInit r_bup_interrupt
	r_cg_bup.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_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_cg_saic_user.c	R_SAIC_Create_UserInit
	r_cg_saic.h	—

付録 C　API 関数

本付録では，コード生成が出力する API 関数について説明します。

C. 1　概　　要

以下に，コード生成が API 関数を出力する際の命名規則を示します。

- マクロ名

  すべて大文字。

  なお，先頭に“数字”が付与されている場合，該当数字（16 進数値）とマクロ値は同値。

- ローカル変数名

  すべて小文字。

- グローバル変数名

  先頭に“g”を付与し，構成単語の先頭のみ大文字。

- グローバル変数へのポインタ名

  先頭に“gp”を付与し，構成単語の先頭のみ大文字。

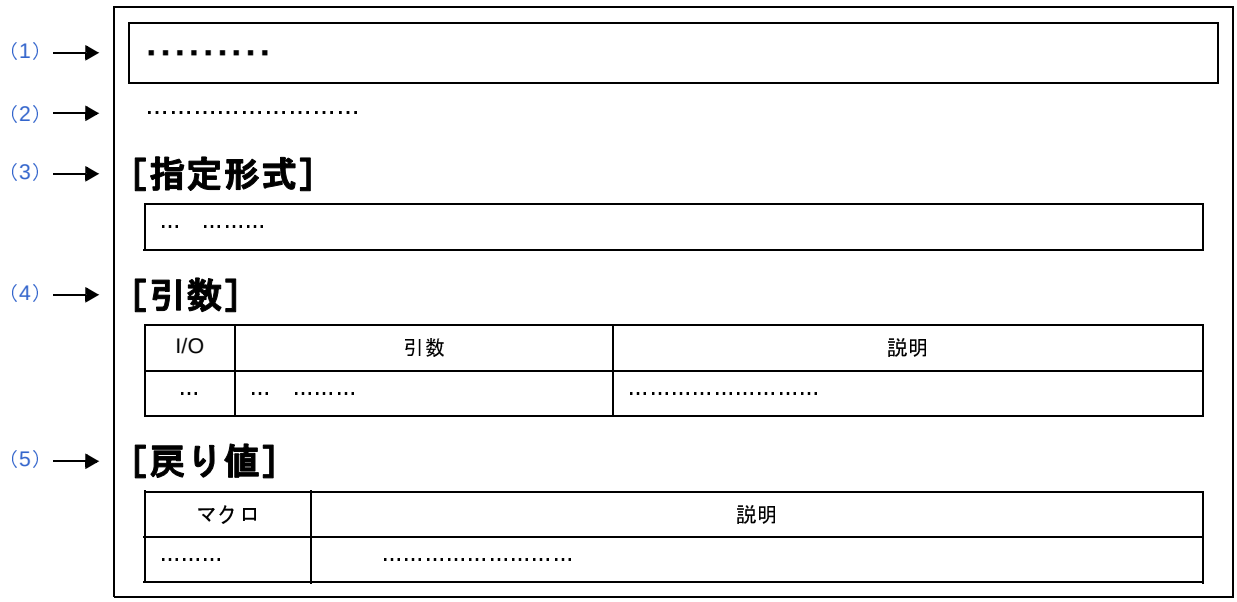
- 列挙指定子 enum の要素名

  すべて大文字。

C. 2　関数リファレンス

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

図 C—1　API 関数の記述フォーマット



(1) 名称

API 関数の名称を示しています。

(2) 機能

API 関数の機能概要を示しています。

(3) [指定形式]

API 関数を C 言語で呼び出す際の記述形式を示しています。

(4) [引数]

API 関数の引数を次の形式で示しています。

I/O	引数	説明
(a)	(b)	(c)

(a) I/O

引数の種類

I … 入力引数

O … 出力引数

(b) 引数

引数のデータ・タイプ

(c) 説明

引数の説明

(5) [戻り値]

API 関数からの戻り値を次の形式で示しています。

マクロ	説明
(a)	(b)

(a) マクロ

戻り値のマクロ

(b) 説明

戻り値の説明

C. 2. 1 共　　通

以下に，コード生成が共通用として出力する API 関数の一覧を示します。

表 C—1 共通用 API 関数

API 関数名	機能概要
hdwinit	各種ハードウエアを制御するうえで必要となる初期化処理を行います。
R_Systeminit	各種周辺機能を制御するうえで必要となる初期化処理を行います。
main	main 関数です。
R_MAIN_UserInit	ユーザ独自の初期化処理を行います。

hdwinit

各種ハードウエアを制御するうえで必要となる初期化処理を行います。

備考 本 API 関数の呼び出しは，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    hdwinit ( void );
```

[引数]

なし

[戻り値]

なし

R_Systeminit

各種周辺機能を制御するうえで必要となる初期化処理を行います。

備考 本 API 関数は，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_UserInit

ユーザ独自の初期化処理を行います。

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

[指定形式]

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

[引数]

なし

[戻り値]

なし

C. 2. 2 クロック発生回路

以下に，コード生成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 C—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	RAM 1 bit 訂正／2 bit 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／周辺ハードウエア・クロックを変更します。
R_CGC_Set_CRCOn	CRC 演算機能を開始します。
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。

[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

RAM 1 bit 訂正／ 2 bit 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，RAM 1 bit 訂正／ 2 bit 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]

```
__interrupt static  void    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]

```
__interrupt static  void    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]

```
__interrupt static  void    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／周辺ハードウエア・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／周辺ハードウエア・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_Set_CRCOn

CRC 演算機能を開始します。

[指定形式]

```
void    R_CGC_Set_CRCOn ( void );
```

[引数]

なし

[戻り値]

なし

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

C. 2. 3 ポート機能

以下に，コード生成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 C―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。

[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

C. 2. 4 タイマ・アレイ・ユニット

以下に，コード生成がタイマ・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 C―4 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_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・レベルの測定幅を獲得します。
R_TAUm_Channeln_Set_SoftwareTriggerOn	ワンショット・パルス出力のためのトリガ（ソフトウエア・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。

[指定形式]

```
void    R_TAUm_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_TAUm_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* 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]

```
__interrupt static  void    r_taum_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_tau*m*_channel*n*_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。

備考 本 API 関数は，タイマ割り込み INTTM*mn*H 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]

```
__interrupt static  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 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，該当機能の種類（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。

[指定形式]

```
void    R_TAUm_Channeln_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 関数の呼び出しは，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_TAUm_Channeln_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 関数の呼び出しは，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。

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，該当機能の種類（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_TAUm_Channeln_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。

[指定形式]

```
void    R_TAUm_Channeln_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 関数の呼び出しは，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_TAUm_Channeln_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 関数の呼び出しは，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_TAUm_Channeln_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*_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 関数の呼び出しにより，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_TAUm_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・レベルの測定幅を獲得します。

[指定形式]

```
#include    "r_cg_macrodriver.h"
void    R_TAUm_Channeln_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	`uint32_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*_Channel*n*_Set_SoftwareTriggerOn

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

[指定形式]

```
void    R_TAUm_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

C. 2. 5 タイマ RJ

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

表 C—5 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_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。

[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]

```
__interrupt static  void    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 関数の呼び出しにより，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 関数の呼び出しは，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。

2. パルス幅の計測中にオーバフロー（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"
void    R_TMR_RJ0_Get_PulseWidth ( uint32_t * const active_width );
```

[引数]

I/O	引数	説明
O	uint32_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

C. 2. 6 タイマ RD

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

表 C—6 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_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。

[指定形式]

```
void    R_TMR_RDn_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_RDn_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]

```
__interrupt static  void    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 関数の呼び出しにより，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_ForcedOutput_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処理を終了します。

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 関数の呼び出しは，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。

2. パルス幅の計測中にオーバフロー（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"
MD_STATUS   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	uint32_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]

マクロ	説明
MD_OK	正常終了

C. 2. 7 タイマ RG

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

表 C―7 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。

[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]

```
__interrupt static  void    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 関数の呼び出しにより，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 関数の呼び出しは，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。

2. パルス幅の計測中にオーバフロー（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"
MD_STATUS   R_TMR_RG0_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	uint32_t * const *active_width*;	TRGIOA 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*;	TRGIOA 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]

マクロ	説明
MD_OK	正常終了

C. 2. 8 16 ビット・タイマ KB

以下に，コード生成が 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 C—8 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_KBm_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_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。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_TMR_KB*m*_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_KBm_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_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]

```
__interrupt static  void    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 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 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 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・タイマ KB 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_ForcedOutput_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_BatchOverwriteRequestOn ( 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

C. 2. 9 16 ビット・タイマ KC0

以下に，コード生成が 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 C—9 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。

[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]

```
__interrupt static  void    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 関数の呼び出しにより，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

C. 2. 10 16 ビット・タイマ KB2

以下に，コード生成が 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 C—10 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_Set_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。

[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_KB2m_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]

```
__interrupt static  void    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_Output_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 TKB0*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 TKB0*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_Output_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 TKB0*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 TKB0*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 TKB0*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 TKB0*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 TKB0*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 TKB0*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 TKB0*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 TKB0*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 TKB0*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*_Set_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_KB2m_Set_BatchOverwriteRequestOn ( 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

C. 2. 11 リアルタイム・クロック

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

表 C—11 リアルタイム・クロック用 API 関数

API 関数名	機能概要
R_RTC_Create	リアルタイム・クロックを制御するうえで必要となる初期化処理を行います。
R_RTC_Create_UserInit	リアルタイム・クロックに関するユーザ独自の初期化処理を行います。
r_rtc_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_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。
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_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_AlarmValue	アラームの発生条件（曜日，時，分）を読み出します。
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（1 Hz）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（1 Hz）の出力を禁止します。

R_RTC_Create

リアルタイム・クロックを制御するうえで必要となる初期化処理を行います。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_RTC_Create_UserInit

リアルタイム・クロックに関するユーザ独自の初期化処理を行います。

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

[指定形式]

```
void    R_RTC_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_rtc_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]

```
__interrupt static  void    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 関数の呼び出しにより，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。

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他の周辺装置（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"
MD_STATUS   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	カウント処理を停止中（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。

[指定形式]

```
#include    "r_cg_macrodriver.h"
#include    "r_cg_rtc.h"
MD_STATUS   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，リアルタイム・クロックのカウント値 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 */
    uint8_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	カウント処理を停止中（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_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"
MD_STATUS   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（読み出し前）
MD_BUSY2	カウント処理を停止中（読み出し後）

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_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_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]

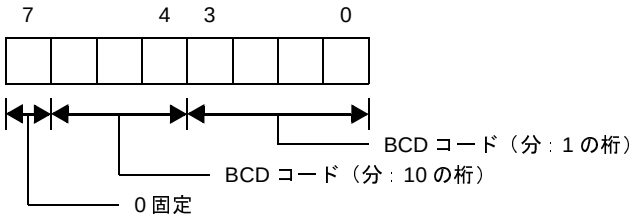
I/O	引数	説明
I	rtc_alarm_value_t *alarm_val*;	アラームの発生条件（曜日，時，分）

備考 以下に，アラームの発生条件 rtc_alarm_value_t の構成を示します。

```
typedef struct {
    uint8_t alarmwm;    /* 分 */
    uint8_t alarmwh;    /* 時 */
    uint8_t alarmww;    /* 曜日 */
} rtc_alarm_value_t;
```

- 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。



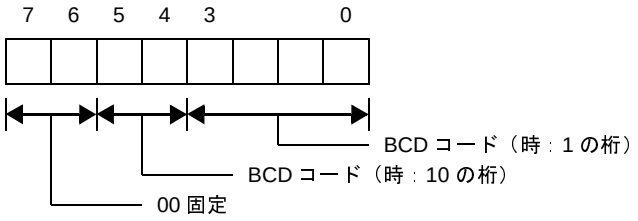
- alarmwh（時）

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。

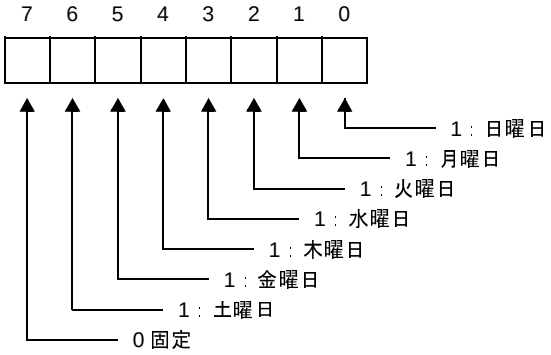
0： 午前

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。



[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 );
```

備考 アラームの発生条件 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。

[引数]

I/O	引数	説明
O	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

[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]

```
static  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（1 Hz）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（1 Hz）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

C. 2. 12 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 C—12 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。

[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 INTFM 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]

```
__interrupt static  void    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

C. 2. 13 12 ビット・インターバル・タイマ

以下に，コード生成が 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 C—13 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_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。

[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]

```
__interrupt static  void    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。

[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。

[指定形式]

```
void    R_IT_Stop ( 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。

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_IT_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

C. 2. 14 8 ビット・インターバル・タイマ

以下に，コード生成が 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 C—14 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_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。

[指定形式]

```
void    R_IT8bitm_Channeln_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_IT8bitm_Channeln_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]

```
__interrupt static  void    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_IT8bitm_Channeln_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_IT8bitm_Channeln_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_IT8bitm_Channeln_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

C. 2. 15 16 ビット・ウエイクアップ・タイマ

以下に，コード生成が 16 ビット・ウエイクアップ・タイマ用として出力する API 関数の一覧を示します。

表 C—15 16 ビット・ウエイクアップ・タイマ用 API 関数

API 関数名	機能概要
R_WUTM_Create	16 ビット・ウエイクアップ・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・ウエイクアップ・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・ウエイクアップ・タイマのカウント処理を開始します。
R_WUTM_Stop	16 ビット・ウエイクアップ・タイマのカウント処理を終了します。
R_WUTM_Set_PowerOff	16 ビット・ウエイクアップ・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・ウエイクアップ・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。

[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・ウエイクアップ・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]

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

[引数]

なし

[戻り値]

なし

R_WUTM_Start

16 ビット・ウエイクアップ・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・ウエイクアップ・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・ウエイクアップ・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・ウエイクアップ・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

C. 2. 16 クロック出力／ブザー出力制御回路

以下に，コード生成がクロック出力／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 C—16 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／ブザー出力を停止します。
R_PCLBUZ_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。

[指定形式]

```
void    R_PCLBUZn_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_PCLBUZn_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_PCLBUZn_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／ブザー出力を停止します。

[指定形式]

```
void    R_PCLBUZn_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_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。

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_PCLBUZ_Set_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

C. 2. 17 ウォッチドッグ・タイマ

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

表 C—17 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_interrupt	インターバル割り込み 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。

[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]

```
__interrupt static  void    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

C. 2. 18 A/D コンバータ

以下に，コード生成が 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 C—18 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_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	A/D 変換結果（10 ビット）を読み出します。
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。

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。

[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]

```
__interrupt static  void    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。

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停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_ADC_Start ( void );
```

[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_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

備考 アナログ電圧の入力端子 ADCHANNEL*n* についての詳細は，ヘッダ・ファイル r_cg_adc.h を参照してください。

[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） ADAVREFM : テスト・モード（AVREFM 入力電圧） ADAVREFP : 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。

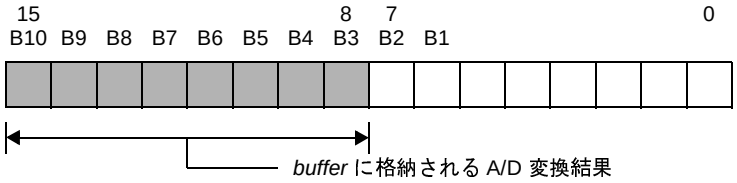
[指定形式]

```
#include    "r_cg_macrodriver.h"
void    R_ADC_Get_Result_8bit ( 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

C. 2. 19 温度センサ

以下に，コード生成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 C―19 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_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。

[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_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

C. 2. 20 24 ビット ΔΣA/D コンバータ

以下に，コード生成が 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 C—20 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_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_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 ΔΣ A/D コンバータを制御するうえで必要となる初期化処理を行います。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_DSADC_Create_UserInit

24 ビット ΔΣ 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_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]

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

[引数]

なし

[戻り値]

なし

R_DSADC_Set_OperationOn

24 ビット ΔΣ A/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 ΔΣ A/D コンバータを動作停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_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 コンバータに対する電荷リセットを行った場合，約 1.4μ 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）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

C. 2. 21 D/A コンバータ

以下に，コード生成が 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 C—21 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_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。

[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_DACn_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_DACn_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_DACn_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 変換値（0x0 ～ 0xFF）

[戻り値]

なし

C. 2. 22 プログラマブル・ゲイン・アンプ

以下に，コード生成がプログラマブル・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 C—22 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_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。

[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

C. 2. 23 コンパレータ

以下に，コード生成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 C—23 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停止します。
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。

[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]

```
__interrupt static  void    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_COMPn_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_COMPn_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_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

C. 2. 24 シリアル・アレイ・ユニット

以下に，コード生成がシリアル・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 C—24 シリアル・アレイ・ユニット用 API 関数

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_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_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。
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。

API 関数名	機能概要
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。
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。

[指定形式]

```
void    R_SAUm_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_SAUm_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_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_SAUm_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_SAUm_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_SAUm_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_UARTn_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]

```
__interrupt static  void    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]

```
__interrupt static  void    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]

```
__interrupt static  void    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_UARTn_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_UARTn_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。

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。

2. 実際の UART 受信は，本 API 関数の呼び出し後，R_UARTn_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_UARTn_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 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_uartn_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 00000*xx*1B：オーバラン・エラー 00000*x*1*x*B：パリティ・エラー 000001*xx*B：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_CSImn_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 通信完了割り込み 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]

```
__interrupt static  void    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_CSImn_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_CSImn_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。

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。

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。

2. 本 API 関数では，1 バイト単位の CSI 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。

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 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。

[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 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，または R_CSImn_Send_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_csimn_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 00000*xx*1B：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_IICmn_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 通信完了割り込み 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 通信完了割り込み INTIIC*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]

```
__interrupt static  void    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_IICmn_Master_Send，および R_IICmn_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_IICmn_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させます。

[指定形式]

```
void    R_IICmn_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_IICmn_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 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。

[指定形式]

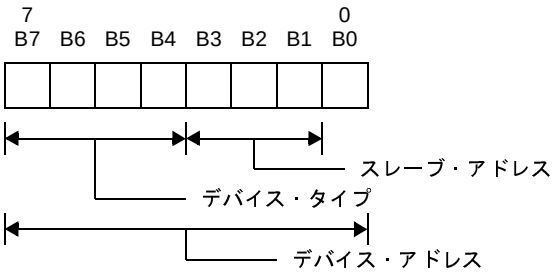
```
#include    "r_cg_macrodriver.h"
void    R_IICmn_Master_Send ( uint8_t adr, uint8_t * const tx_buf, uint16_t tx_num );
```

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 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。

[指定形式]

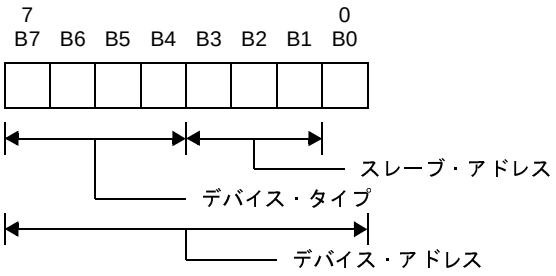
```
#include    "r_cg_macrodriver.h"
void    R_IICmn_Master_Receive ( uint8_t adr, uint8_t * const rx_buf, uint16_t rx_num );
```

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*;	デバイス・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 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 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_iicmn_interrupt のコールバック・ルーチン（R_IICmn_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 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 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_iicmn_interrupt のコールバック・ルーチン（R_IICmn_Master_Receive の引数 *rx_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_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

パリティ・エラー（ACK 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_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

[戻り値]

なし

C. 2. 25 シリアル・アレイ・ユニット 4（DALI/UART4）

以下に，コード生成がシリアル・アレイ・ユニット 4（DALI/UART4）用として出力する API 関数の一覧を示します。

表 C—25 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（DALI/UART4）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（DALI/UART4）を制御するうえで必要となる初期化処理を行います。

[指定形式]

```
void    R_DALIn_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]

```
__interrupt static  void    r_dalin_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]

```
__interrupt static  void    r_dalin_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]

```
__interrupt static  void    r_dalin_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_DALIn_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_DALIn_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。

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。

2. 実際の DALI 受信は，本 API 関数の呼び出し後，R_DALIn_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_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 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 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_dalin_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_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。

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 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 00000*xx*1B：オーバラン・エラー 00000*x*1*x*B：パリティ・エラー 000001*xx*B：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_dalin_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

C. 2. 26 アシンクロナス・シリアル・インタフェース LIN-UART（UARTF）

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

表 C—26 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（UARTF）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（UARTF）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（UARTF）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（UARTF）を制御するうえで必要となる初期化処理を行います。

[指定形式]

```
void    R_UARTFn_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（UARTF）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_UARTFn_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 INTLT 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]

```
__interrupt static  void    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]

```
__interrupt static  void    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]

```
__interrupt static  void    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_UARTFn_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_UARTFn_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（UARTF）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（UARTF）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_UARTFn_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。

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。

3. アシンクロナス・シリアル・インタフェース LIN-UART（UARTF）を拡張ビット・モードで使用する場合，引数 *tx_buf* で指定された場合には，送信するデータを以下の形式で格納します。

“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]

```
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。

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。

3. アシンクロナス・シリアル・インタフェース LIN-UART（UARTF）を拡張ビット・モードで使用する場合，引数 *rx_buf* で指定された場合には，受信したデータが以下の形式で格納されます。

“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]

```
MD_STATUS   R_UARTFn_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（UARTF）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_UARTFn_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（UARTF）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_UARTFn_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 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 00000*xx*1B：オーバラン・エラー 00000*x*1*x*B：パリティ・エラー 000001*xx*B：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_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

C. 2. 27 シリアル・インタフェース IICA

以下に，コード生成がシリアル・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 C—27 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_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_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。
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エ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エ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エ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エ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。

[指定形式]

```
void    R_IICAn_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_IICAn_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]

```
__interrupt static  void    r_iican_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。

[指定形式]

```
void    R_IICAn_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_IICAn_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*_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（IICA コントロール・レジスタ *n*：IICCTL*n* など）への書き込みは無視されます。

[指定形式]

```
void    R_IICAn_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

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR1	バス通信状態
MD_ERROR2	バス未解放状態

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_STATUS   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

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR1	バス通信状態
MD_ERROR2	バス未解放状態

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 本 API 関数は，IICA マスタ送信完了割り込み 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_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 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。

[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

[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_iican_callback_slave_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_iican_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

[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_iican_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エイクアップ機能の動作を許可します。

[指定形式]

```
void    R_IICAn_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エイクアップ機能の動作を禁止します。

[指定形式]

```
void    R_IICAn_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エイクアップ機能の動作を許可します。

[指定形式]

```
void    R_IICAn_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エイクアップ機能の動作を禁止します。

[指定形式]

```
void    R_IICAn_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

C. 2. 28 LCD コントローラ／ドライバ

以下に，コード生成が 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 C―28 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_Create

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

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_LCD_Create_UserInit

LCD コントローラ／ドライバに関するユーザ独自の初期化処理を行います。

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

[指定形式]

```
void    R_LCD_Create_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

r_lcd_interrupt

LCD フレーム割り込み INTLCD の発生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]

```
__interrupt static  void    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可能状態にします。

[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。

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

C. 2. 29 サウンド・ジェネレータ

以下に，コード生成がサウンド・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 C—29 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。
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。

[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]

```
__interrupt static  void    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停止状態にします。

[指定形式]

```
void    R_SG_Stop ( void );
```

[引数]

なし

[戻り値]

なし

C. 2. 30 DMA コントローラ

以下に，コード生成が 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 C—30 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。

[指定形式]

```
void    R_DMACn_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_DMACn_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。

[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]

```
__interrupt static  void    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_DMACn_Start ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_DMAC*n*_Stop

チャネル *n* 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。

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_DMACn_Stop ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

R_DMAC*n*_Set_SoftwareTriggerOn

DMA 転送を開始します。

[指定形式]

```
void    R_DMACn_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

C. 2. 31 DTC

以下に，コード生成が DTC 用として出力する API 関数の一覧を示します。

表 C—31 DTC 用 API 関数

API 関数名	機能概要
R_DTC_Create	DTC 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	DTC に関するユーザ独自の初期化処理を行います。
R_DTCn_Start	DTC を動作可能状態にします。
R_DTCn_Stop	DTC を動作停止状態にします。
R_DTC_Set_PowerOff	DTC に対するクロック供給を停止します。

R_DTC_Create

DTC を制御するうえで必要となる初期化処理を行います。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_DTC_Create_UserInit

DTC 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

DTC を動作可能状態にします。

[指定形式]

```
void    R_DTCn_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

DTC を動作停止状態にします。

[指定形式]

```
void    R_DTCn_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

DTC 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，DTC 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

C. 2. 32 イベントリンクコントローラ（ELC）

以下に，コード生成がイベントリンクコントローラ（ELC）用として出力する API 関数の一覧を示します。

表 C—32 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（ELC）を制御するうえで必要となる初期化処理を行います。
R_ELC_Create_UserInit	イベントリンクコントローラ（ELC）に関するユーザ独自の初期化処理を行います。
R_ELC_Stop	イベントリンクコントローラ（ELC）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（ELC）を制御するうえで必要となる初期化処理を行います。

[指定形式]

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

[引数]

なし

[戻り値]

なし

R_ELC_Create_UserInit

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

備考 本 API 関数は，R_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（ELC）を動作停止状態にします。

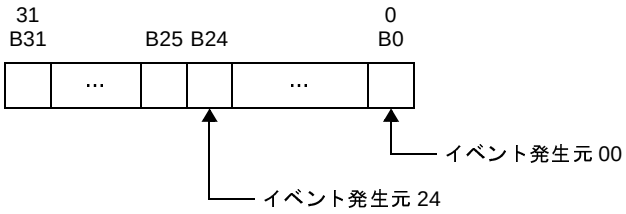
[指定形式]

```
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

C. 2. 33 割り込み機能

以下に，コード生成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 C—33 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_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。

[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]

```
__interrupt static  void    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_INTCn_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_INTCn_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]

```
__interrupt static  void    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_INTCLRn_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_INTCLRn_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

C. 2. 34 キー割り込み機能

以下に，コード生成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 C―34 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。

[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]

```
__interrupt static  void    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

C. 2. 35 電圧検出回路

以下に，コード生成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 C—35 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_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。

[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]

```
__interrupt static  void    r_lvd_interrupt ( 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

C. 2. 36 バッテリ・バックアップ機能

以下に，コード生成がバッテリ・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 C—36 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。

[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]

```
__interrupt static  void    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

C. 2. 37 発振停止検出回路

以下に，コード生成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 C―37 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_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。

[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]

```
__interrupt static  void    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，発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

C. 2. 38 SPI インタフェース

以下に，コード生成が 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 C—38 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_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。

[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

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2013.09.01	―	初版発行

CubeSuite+ V2.01.00　ユーザーズマニュアル
RL78 設計編

発行年月日　2013年 9月 1日　　Rev.1.00

発行　　ルネサス エレクトロニクス株式会社
　　〒211-8668　神奈川県川崎市中原区下沼部 1753

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口 http://www.renesas.com

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

ルネサス エレクトロニクス販売株式会社　〒100-0004　千代田区大手町2-6-2（日本ビル） (03)5201-5307

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

CubeSuite+ V2.01.00

RENESAS

ルネサス エレクトロニクス株式会社

R20UT2684JJ0100