User's Manual

CubeSuite+ V1.03.00

統合開発環境

ユーザーズマニュアル　78K0R 設計編

対象デバイス

78K0Rマイクロコントローラ

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　2012.09

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，78K0R マイクロコントローラ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，78K0R マイクロコントローラの統合開発環境（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	R20UT2133J	R20UT2133E
	V850 設計編	R20UT2134J	R20UT2134E
	RL78 設計編	R20UT2136J	R20UT2136E
	78K0R 設計編	このマニュアル	R20UT2137E
	78K0 設計編	R20UT2138J	R20UT2138E
	RX コーディング編	R20UT0767J	R20UT0767E
	V850 コーディング編	R20UT0553J	R20UT0553E
	コーディング編（CX コンパイラ）	R20UT2139J	R20UT2139E
	RL78, 78K0R コーディング編	R20UT2140J	R20UT2140E
	78K0 コーディング編	R20UT2141J	R20UT2141E
	RX ビルド編	R20UT0768J	R20UT0768E
	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
	RX デバッグ編	R20UT2175J	R20UT2175E
	V850 デバッグ編	R20UT2144J	R20UT2144E
	RL78 デバッグ編	R20UT2145J	R20UT2145E
	78K0R デバッグ編	R20UT0732J	R20UT0732E
	78K0 デバッグ編	R20UT0731J	R20UT0731E
	解析編	R20UT2146J	R20UT2146E
	メッセージ編	R20UT2147J	R20UT2147E

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，I/O ポートなど）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ができます。

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。

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

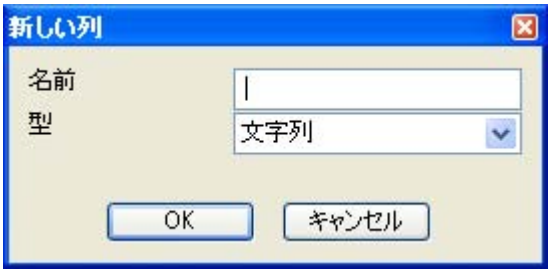
備考 表示順序の変更は，端子配置表の左上に設けられた ボタンをクリックすることによりオープンする列の選択 ダイアログの表示項目選択エリアに表示されている項目をドラッグしたのち，端子配置表の移動先にドロップすることでも，表示項目の表示順序を変更することができます。

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

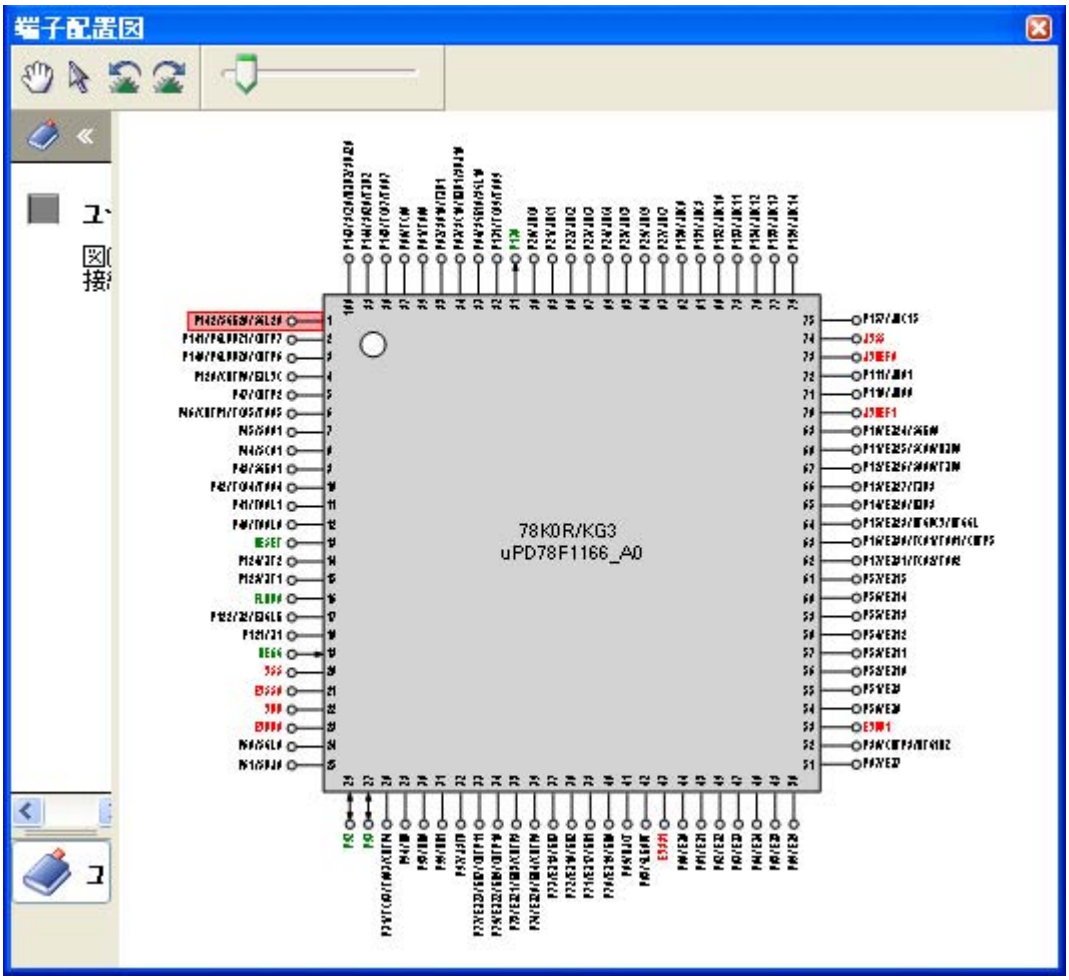
備考 削除可能な列は，新しい列 ダイアログでユーザが独自に追加した列に限られます。

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



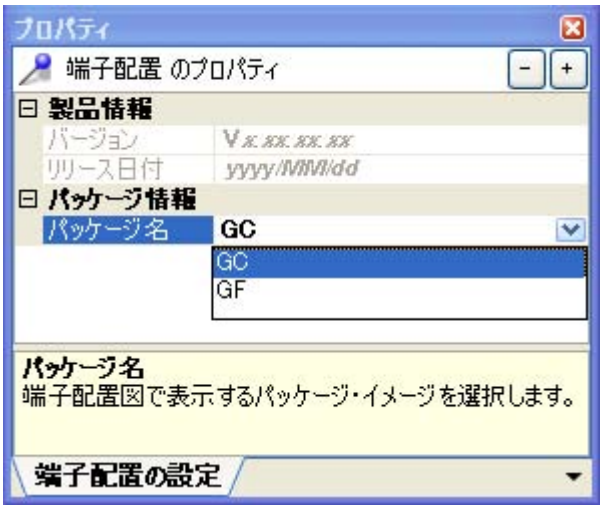
備考 プロパティ パネルの［端子配置の設定］タブでパッケージ名に“BGA”を選択している場合，端子配置図 パネルをオープンすることができません。

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］からオープンするカラー・パレット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

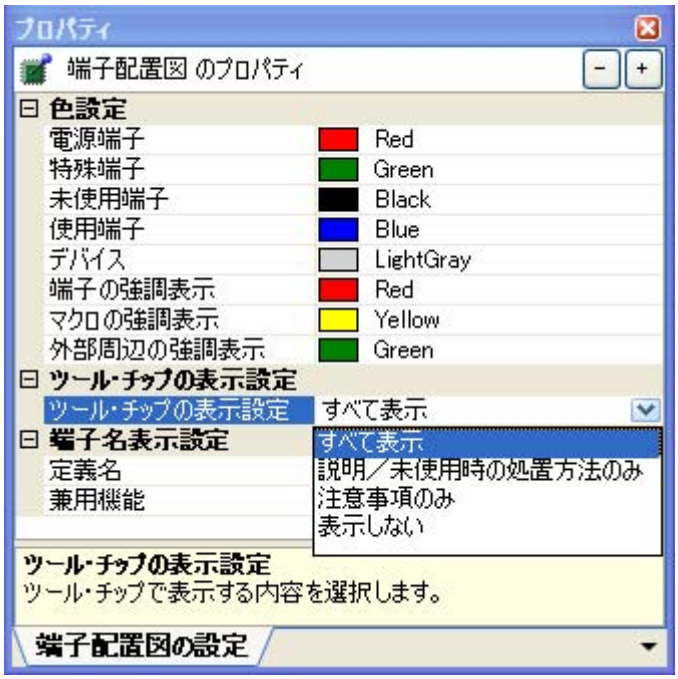
設定対象	概要
電源端子	電源端子（用途が電源に限定されている端子）の表示色を選択します。
特殊端子	特殊端子（用途が規定されている端子）の表示色を選択します。
未使用端子	未使用端子（端子配置表 パネルにおいて，用途が未設定の兼用端子）の表示色を選択します。
使用端子	使用端子（端子配置表 パネルにおいて，用途が設定済みの兼用端子）の表示色を選択します。
デバイス	マイクロコントローラ本体部の表示色を選択します。
端子の強調表示	端子配置表 パネルの［端子番号］タブで選択された項目に対応した端子の背景色を選択します。
マクロの強調表示	端子配置表 パネルの［マクロ］タブで選択された項目に対応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

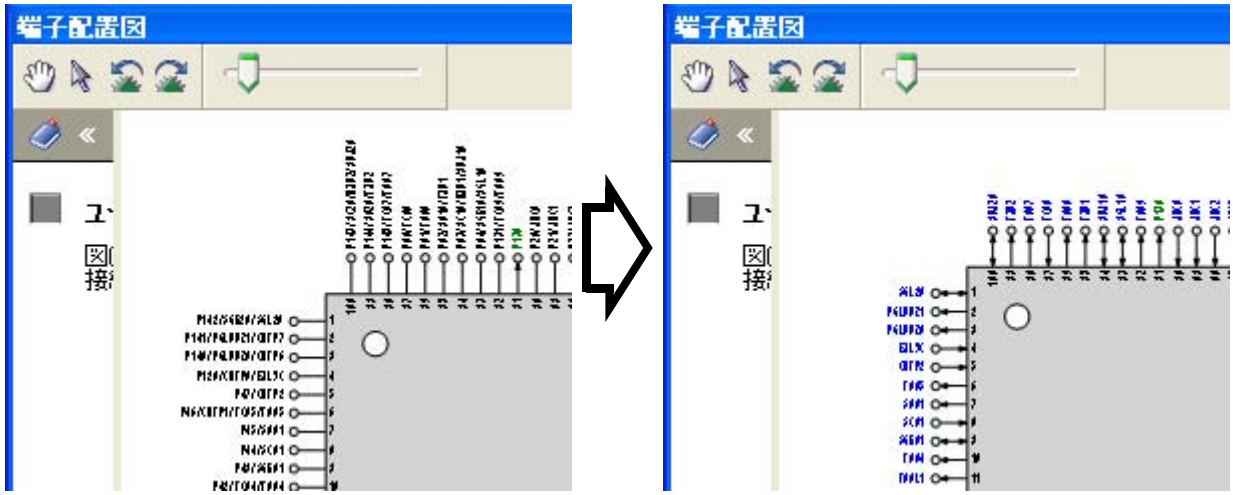
2.4　情報の記述

「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。

3.1　概　　要

コード生成は，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，I/O ポートなど）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）コード生成 パネルのオープン

周辺機能（クロック発生回路，I/O ポートなど）を制御するうえで必要な情報を設定するためのコード生成 パネルをオープンします。

（4）情報の設定

コード生成 パネルで周辺機能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

コード生成 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。

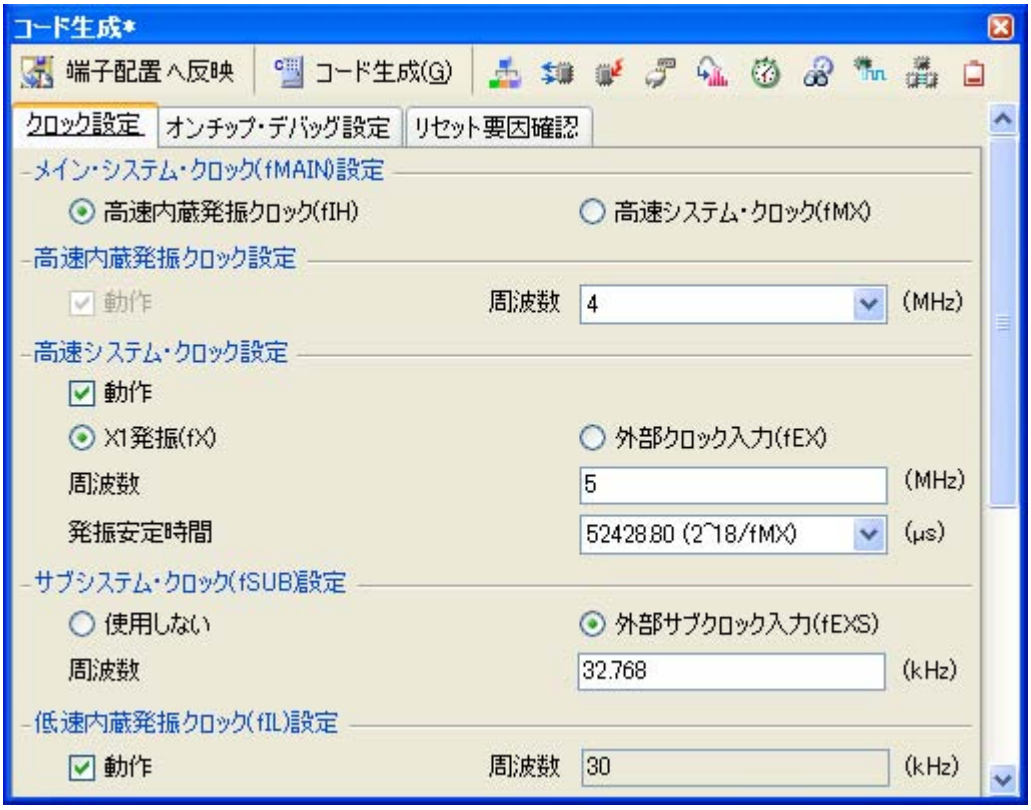
備考 “プロジェクトの保存”についての詳細は，「CubeSuite+ 統合開発環境 ユーザーズマニュアル 起動編」を参照してください。

3.2 コード生成 パネルのオープン

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，I/O ポートなど）を制御するうえで必要な情報を設定するためのコード生成 パネルをオープンします。

なお，コード生成 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）］→周辺機能ノード“［システム］，［ポート］など”をダブルクリックすることにより行います。

図 3—1 コード生成 パネルのオープン



備考 コード生成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“［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 コード生成 パネルのオープン」でオープンしたコード生成 パネルの情報設定エリアで周辺機能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 コード生成 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，コード生成 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

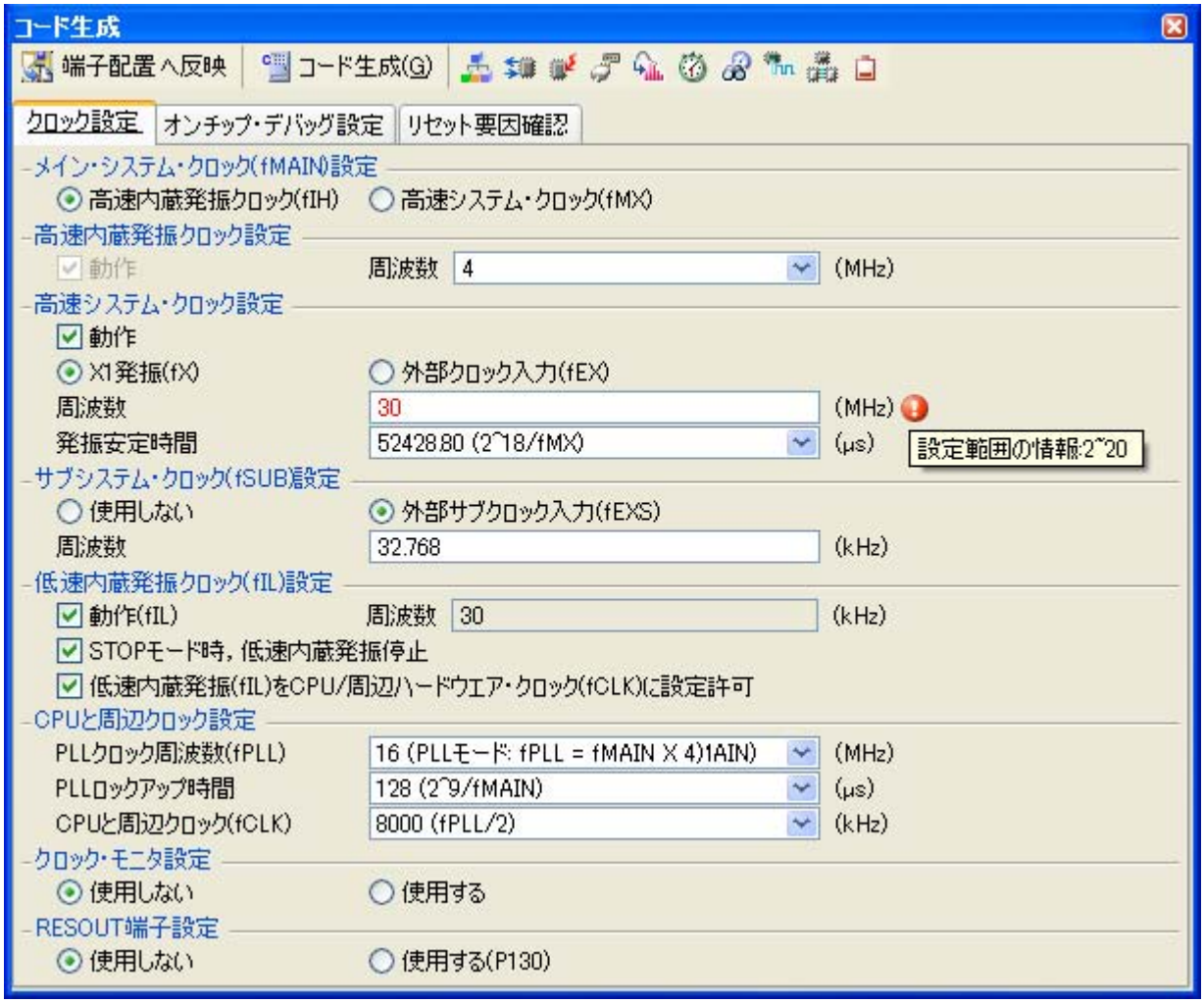
進数表記	概要
10 進数	1 ～ 9 の数字で始まり 0 ～ 9 の数字が続く数値，および 0
16 進数	0x で始まり 0 ～ 9 の数字，および a ～ f の英字が続く数値 （英字の大文字／小文字については，不問）

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，コード生成 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

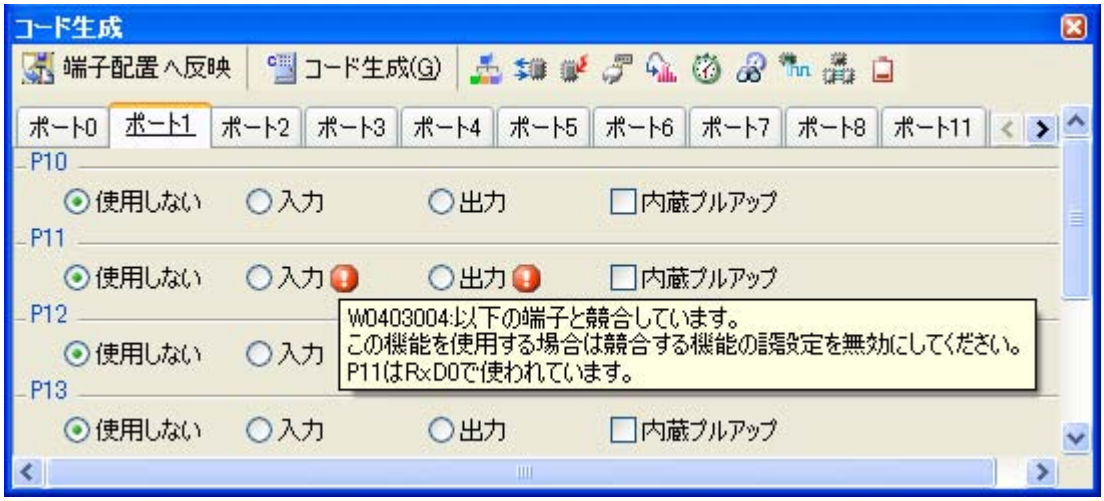


3. 3. 3　端子の競合に対するアイコン表示

コード生成では，コード生成 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

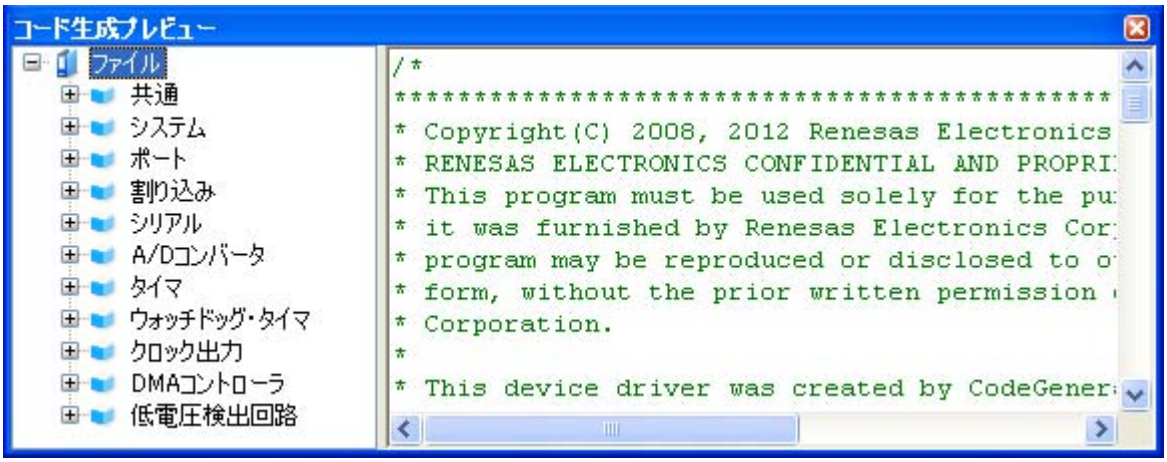


3.4　ソース・コードの確認

「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，［表示］メニュー→［コード生成プレビュー］を選択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. コード生成プレビュー 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

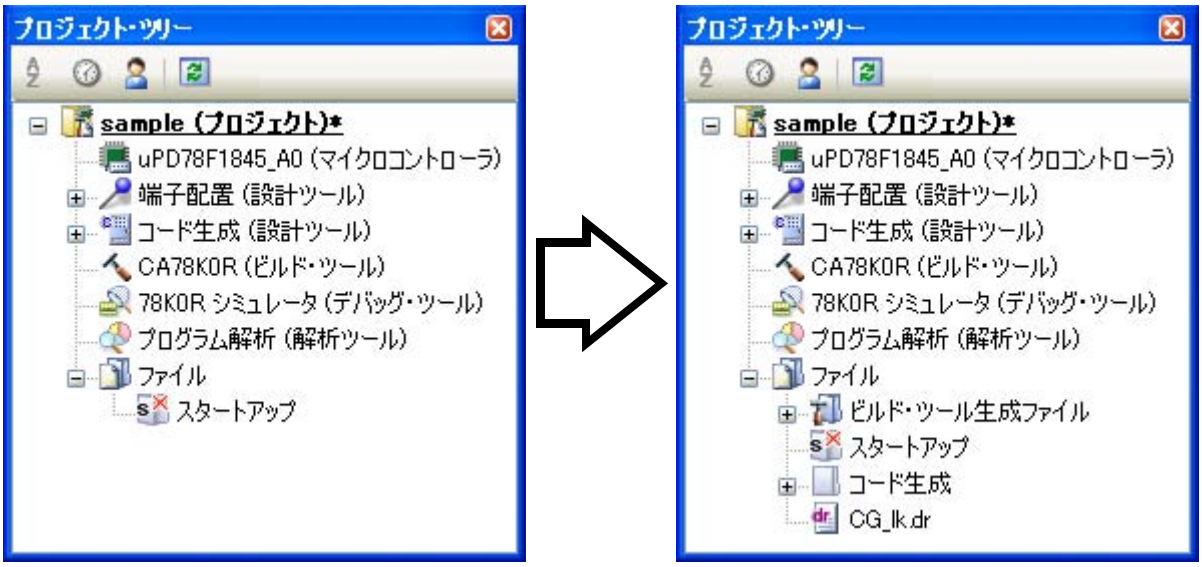
3. コード生成プレビュー パネル内でソース・コードを編集することはできません。

4. 一部の API 関数（シリアル・アレイ・ユニット用 API 関数など）については，ソース・コードの出力時（コード生成 パネルの コード生成(G) ボタンをクリックした際）にレジスタ値 SFR などが計算され確定するものがあります。このため，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

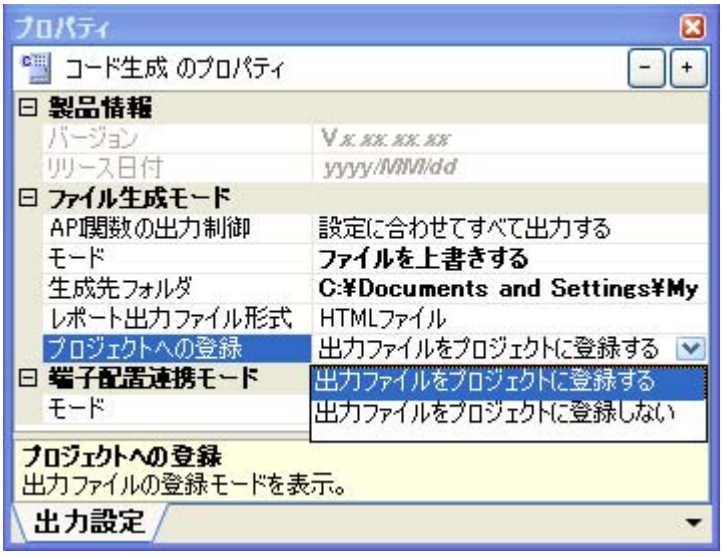
コード生成 パネルの コード生成(G) ボタンをクリックし，ソース・コード（デバイス・ドライバ・プログラム）を出力します。なお，ソース・コードの出力先は，プロパティ パネルの［出力設定］タブ→［生成先フォルダ］で指定されたフォルダとなります。

図 3—5　ソース・コードの出力



備考　コード生成(G) 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［出力設定］タブ→［プロジェクトへの登録］で“出力ファイルをプロジェクトに登録する”を指定する必要があります。

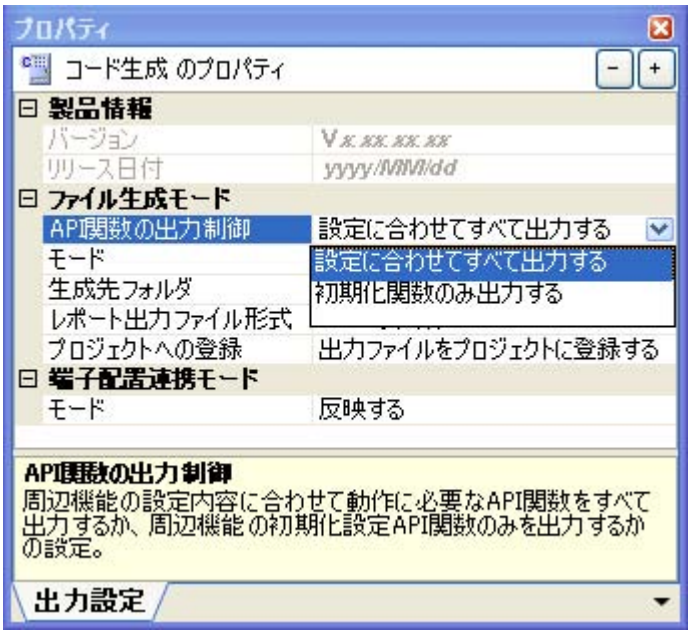
図 3—6　登録有無の設定



3. 5. 1　出力有無の設定

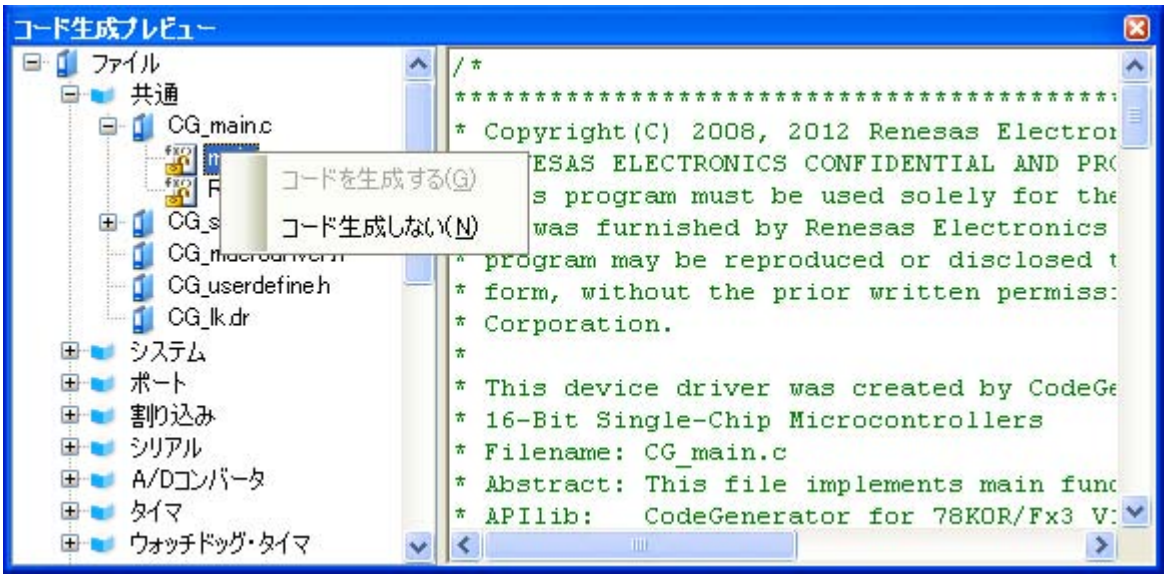
コード生成では，プロパティ パネルの［出力設定］タブ→［API 関数の出力制御］で“設定に合わせてすべて出力する／初期化関数のみ出力する”を選択することにより，出力する API 関数の種類（全 API 関数，初期化用 API 関数のみ）を設定することができます。

図 3—7　出力有無の設定



また，コード生成では，コード生成プレビュー パネルの API 関数名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・メニューから“コードを生成する／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，コード生成プレビュー パネルの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・コードは，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・コードは，出力されます。
	該当 API 関数のソース・コードは，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，コード生成プレビュー パネルのファイル名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。

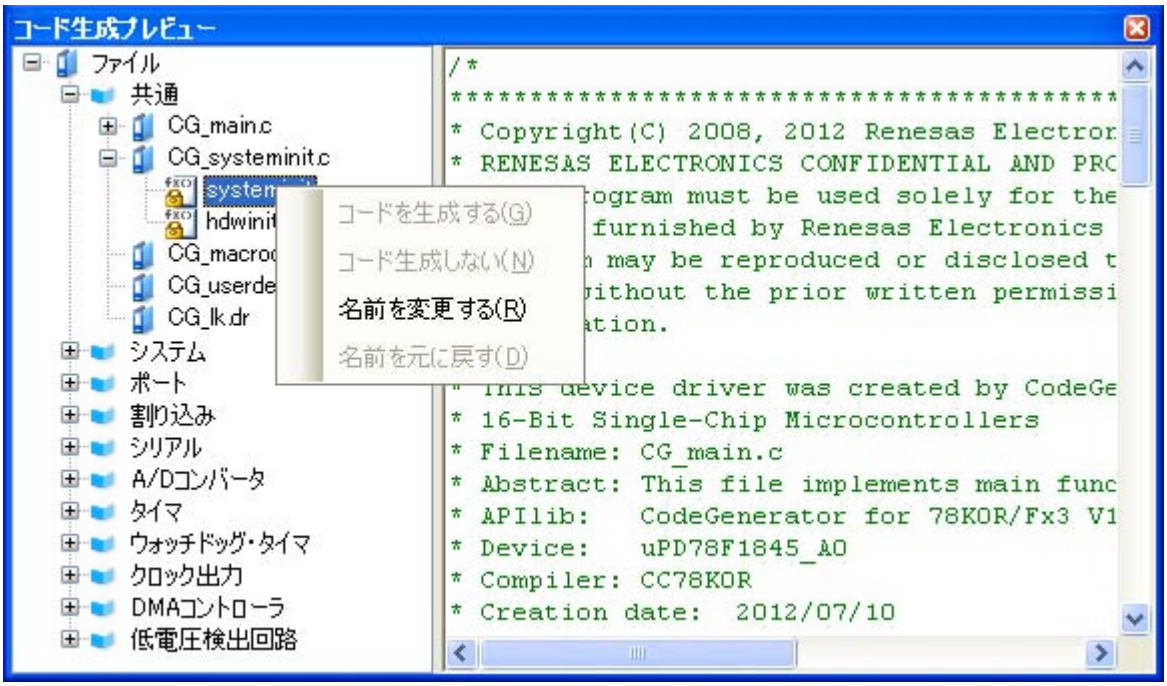
図 3—9 ファイル名の変更

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，コード生成プレビュー パネルの API 関数名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

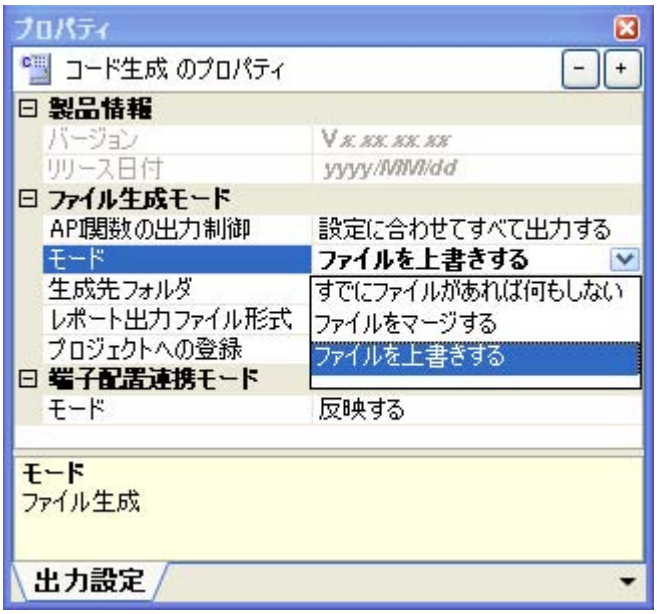


備考 コード生成が規定しているデフォルト API 関数名に戻す際には，コンテキスト・メニューから“名前を元に戻す”を選択します。

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［出力設定］タブ→［モード］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



備考 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

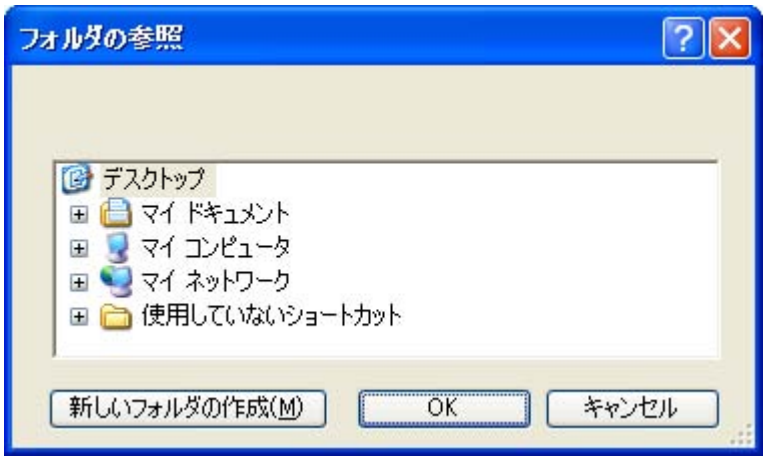
出力モード	概要
すでにファイルがあれば何もしない	同一のファイル名を有するファイルが既存していた場合，該当ファイルの出力を行いません。
ファイルをマージする	同一のファイル名を有するファイルが既存していた場合，該当ファイルをマージします。 なお，マージする部位については， /* Start user code …. Do not edit comment generated here */ から /* End user code. Do not edit comment generated here */ で囲まれた部位に限られます。
ファイルを上書きする	同一のファイル名を有するファイルが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

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［出力設定］タブ→［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。

なお，出力先の変更は，［生成先フォルダ］の［...］ボタンをクリックすることによりオープンするフォルダの参照 ダイアログで行います。

図 3—12　出力先の変更



3.6 レポート・ファイルの出力

コード生成 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［出力設定］タブ→［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，“macro”，および“function”に規定されています。

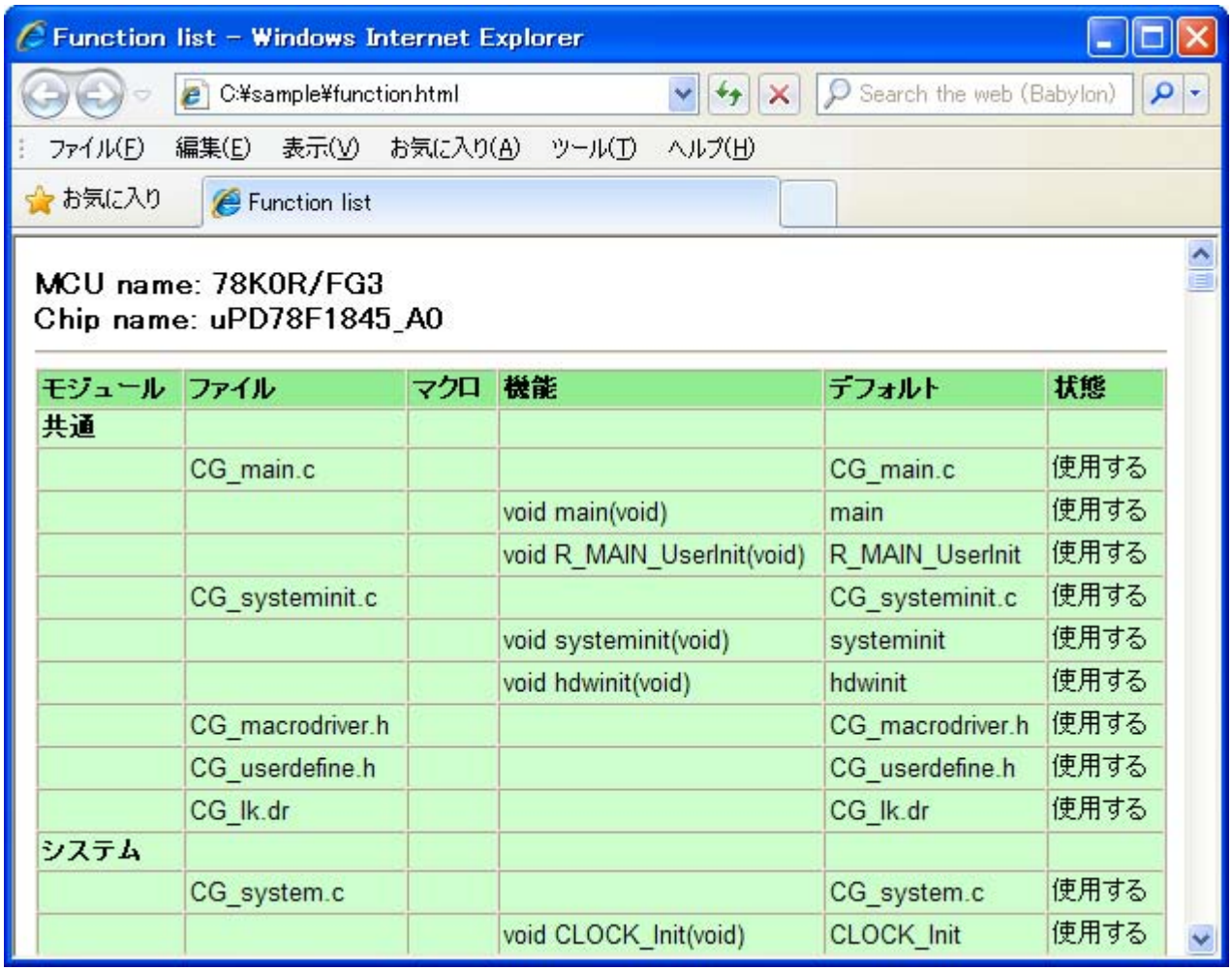
表 3—6 レポート・ファイルの出力

ファイル名	概要
macro	コード生成を用いて設定した情報を保持したファイル
function	ソース・コードに関する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”となります。

図 3—13 レポート・ファイル macro の出力例

図 3—14 レポート・ファイル function の出力例



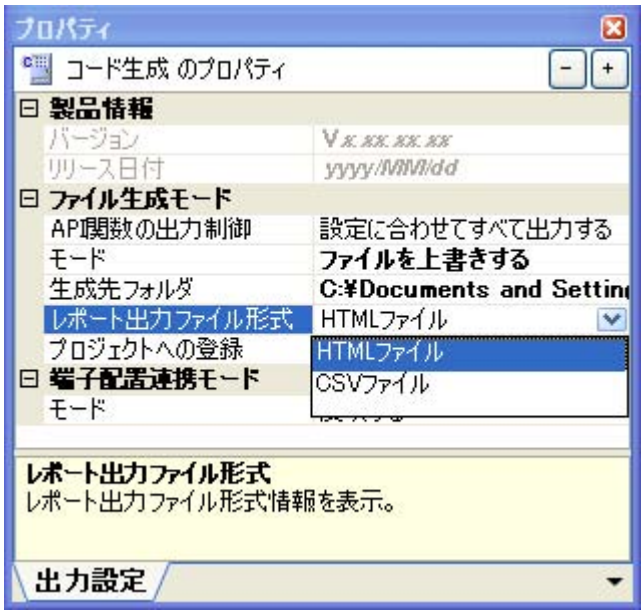
MCU name: 78K0R/FG3
Chip name: uPD78F1845_A0

モジュール	ファイル	マクロ	機能	デフォルト	状態
共通					
	CG_main.c			CG_main.c	使用する
			void main(void)	main	使用する
			void R_MAIN_UserInit(void)	R_MAIN_UserInit	使用する
	CG_systeminit.c			CG_systeminit.c	使用する
			void systeminit(void)	systeminit	使用する
			void hdwinit(void)	hdwinit	使用する
	CG_macrodriver.h			CG_macrodriver.h	使用する
	CG_userdefine.h			CG_userdefine.h	使用する
	CG_lk.dr			CG_lk.dr	使用する
システム					
	CG_system.c			CG_system.c	使用する
			void CLOCK_Init(void)	CLOCK_Init	使用する

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［出力設定］タブ→［レポート出力ファイル形式］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 出力形式の選択は，以下の 2 種類から行います。

表 3—7 ソース・コードの出力モード

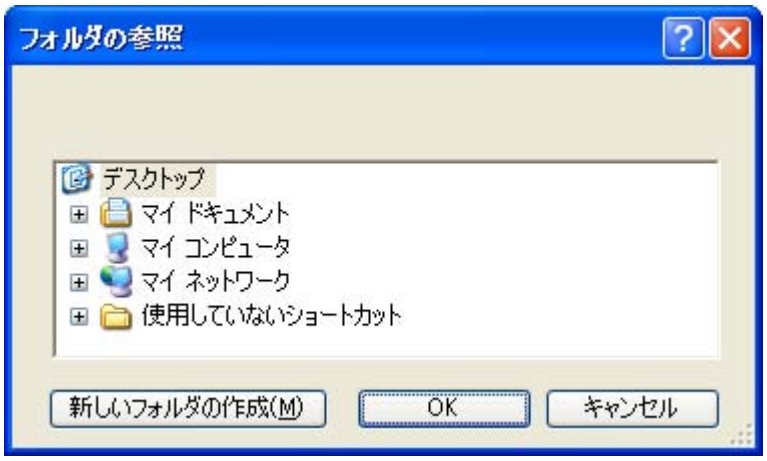
出力形式	概要
HTML ファイル	HTML 形式でレポート・ファイルを出力します。
CSV ファイル	CSV 形式でレポート・ファイルを出力します。

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［出力設定］タブ→［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。

なお，出力先の変更は，［生成先フォルダ］の［...］ボタンをクリックすることによりオープンするフォルダの参照 ダイアログで行います。

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。

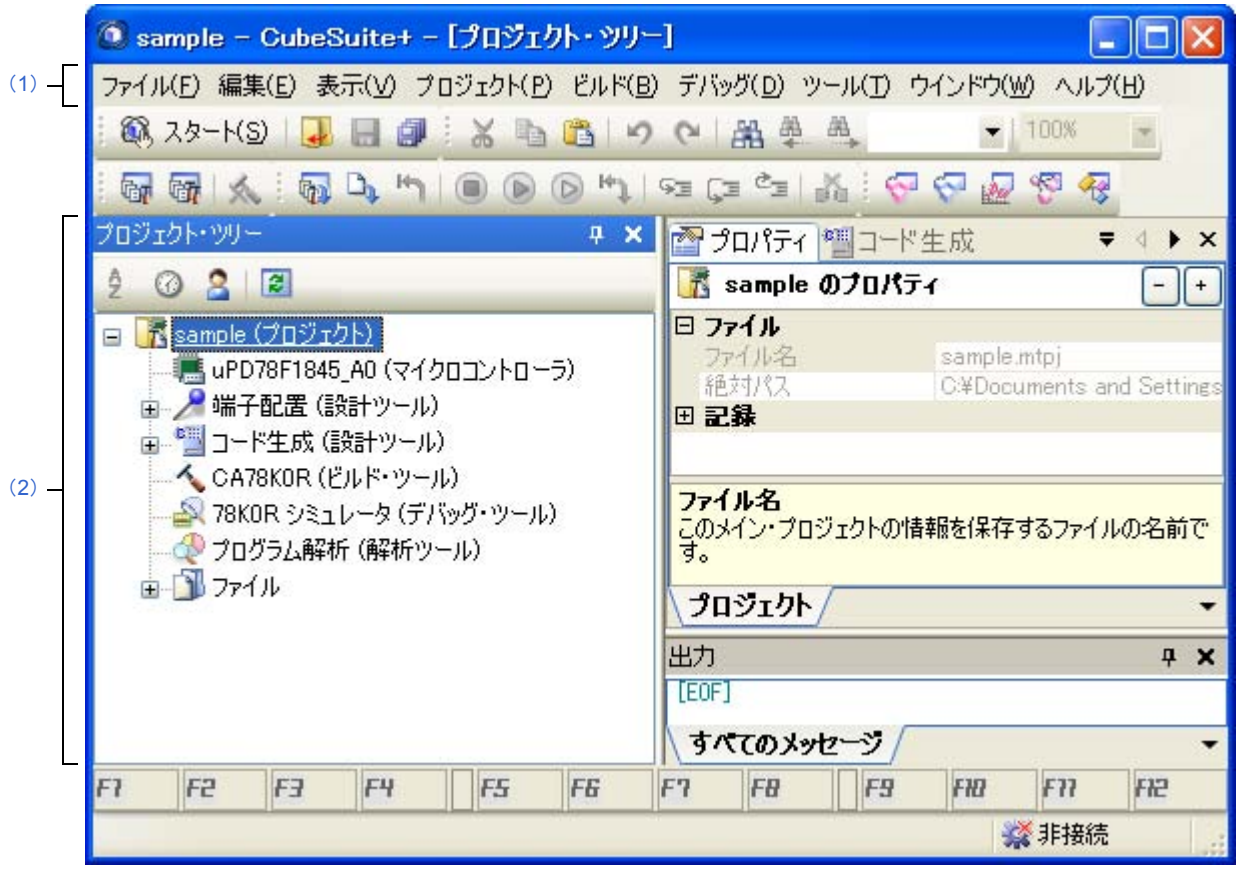
表 A—1　ウインドウ／パネル／ダイアログの一覧

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，コード生成 パネルでクリックした周辺機能ボタン，コード生成プレビュー パネルで選択したファイルの種類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。
コード生成 パネル	マイクロコントローラが提供している周辺機能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	コード生成 パネルの コード生成(G) ボタンをクリックした際に出力されるソース・コード（デバイス・ドライバ・プログラム）の出力有無を API 関数単位で確認／設定するとともに，コード生成 パネルで設定した情報に応じたソース・コードの確認を行います。
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。
フォルダの参照 ダイアログ	ファイル（ソース・コード，レポート・ファイルなど）の出力先を設定します。
名前を付けて保存 ダイアログ	ファイル（レポート・ファイルなど）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保存します。
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保存します。
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	コード生成 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［出力設定］タブ→［レポート出力ファイル形式］で選択された形式（HTML 形式，または CSV 形式）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［出力設定］タブ→［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)［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	コード生成 パネル専用部分 コード生成 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。

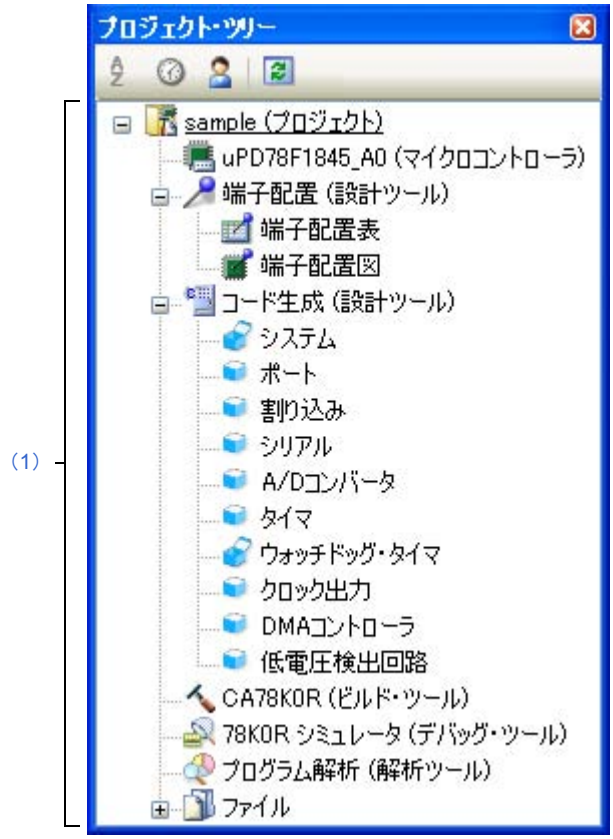
本エリアについての詳細は，以下を参照してください。

- 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
- コード生成 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］
- ［［ヘルプ］メニュー（プロジェクト・ツリー パネル専用部分）］
- ［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，以下に示した周辺機能ノード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，該当周辺機能ノードが表示されません。

システム	マイクロコントローラが提供しているクロック発生回路の機能，オンチップ・デバッグ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［システム］をオープンします。
外部バス	マイクロコントローラが提供している外部バス・インタフェースの機能（外部バスを内蔵 ROM，RAM，SFR 以外の領域に接続する機能）を制御するうえで必要な情報を設定するための［外部バス］をオープンします。
ポート	マイクロコントローラが提供しているポートの機能を制御するうえで必要な情報を設定するための［ポート］をオープンします。
割り込み	マイクロコントローラが提供している割り込み／キー割り込みの機能を制御するうえで必要な情報を設定するための［割り込み］をオープンします。
シリアル	マイクロコントローラが提供しているシリアル・アレイ・ユニット，およびシリアル・インタフェースの機能を制御するうえで必要な情報を設定するための［シリアル］をオープンします。
オペアンプ	マイクロコントローラが提供しているオペアンプの機能を制御するうえで必要な情報を設定するための［オペアンプ］をオープンします。
コンパレータ／ PG アンプ	マイクロコントローラが提供しているコンパレータ／プログラマブル・ゲイン・アンプの機能を制御するうえで必要な情報を設定するための［コンパレータ／ PG アンプ］をオープンします。
A/D コンバータ	マイクロコントローラが提供している A/D コンバータの機能を制御するうえで必要な情報を設定するための［A/D コンバータ］をオープンします。
D/A コンバータ	マイクロコントローラが提供している D/A コンバータの機能を制御するうえで必要な情報を設定するための［D/A コンバータ］をオープンします。
タイマ	マイクロコントローラが提供しているタイマ・アレイ・ユニットの機能を制御するうえで必要な情報を設定するための［タイマ］をオープンします。

ウォッチドッグ・タイマ	マイクロコントローラが提供しているウォッチドッグ・タイマの機能を制御するうえで必要な情報を設定するための［ウォッチドッグ・タイマ］をオープンします。
リアルタイム・カウンタ	マイクロコントローラが提供しているリアルタイム・カウンタの機能を制御するうえで必要な情報を設定するための［リアルタイム・カウンタ］をオープンします。
クロック出力	マイクロコントローラが提供しているクロック出力制御回路の機能を制御するうえで必要な情報を設定するための［クロック出力］をオープンします。
クロック出力／ブザー出力	マイクロコントローラが提供しているクロック出力／ブザー出力制御回路の機能を制御するうえで必要な情報を設定するための［クロック出力／ブザー出力］をオープンします。
LCD コントローラ／ドライバ	マイクロコントローラが提供している LCD コントローラ／ドライバの機能を制御するうえで必要な情報を設定するための［LCD コントローラ／ドライバ］をオープンします。
DMA コントローラ	マイクロコントローラが提供している DMA コントローラの機能を制御するうえで必要な情報を設定するための［DMA コントローラ］をオープンします。
低電圧検出回路	マイクロコントローラが提供している低電圧検出回路の機能を制御するうえで必要な情報を設定するための［低電圧検出回路］をオープンします。

（c）アイコン

周辺機能ノードの各文字列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。

	該当コード生成 パネルに対する操作を実施済み。
	該当コード生成 パネルに対する操作が未実施。
 , 	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。

[［ヘルプ］メニュー（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 パネルのヘルプを開く	本パネルのヘルプを表示します。

[コンテキスト・メニュー]

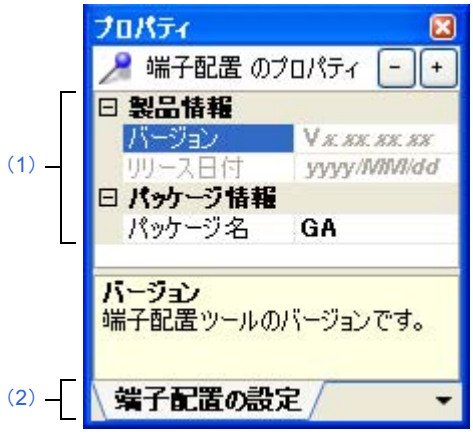
マウスを右クリックすることにより表示されるコンテキスト・メニューは，以下のとおりです。

リセット時の設定に戻す	選択された周辺機能ノードに対応した情報をデフォルトの状態に戻します。
プロパティ	選択されたノード（[端子配置（設計ツール）]，[端子配置表]，[端子配置図]，[コード生成（設計ツール）]，周辺機能ノード“[システム]，[ポート]など”）に対応した情報を保持したプロパティ 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，コード生成 パネルでクリックした周辺機能ボタン，コード生成プレビュー パネルで選択したファイルの種類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（[端子配置（設計ツール）] 選択）



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]
- [[編集] メニュー（プロパティ パネル専用部分）]
- [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，ノード（[端子配置（設計ツール）]，[端子配置表]，[端子配置図]，[コード生成（設計ツール）]，周辺機能ノード“[システム]，[ポート] など”）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，ノード（[端子配置（設計ツール）]，[端子配置表]，[端子配置図]，[コード生成（設計ツール）]，周辺機能ノード“[システム]，[ポート] など”）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
- コード生成プレビュー パネルにおいて，ファイルを選択したのち，[表示] メニュー→ [プロパティ] を選択
- コード生成プレビュー パネルにおいて，ファイルを選択したのち，コンテキスト・メニューから [プロパティ] を選択

備考 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 パネルのノード（[端子配置（設計ツール）]，[端子配置表]，[端子配置図]，[コード生成（設計ツール）]，周辺機能ノード“[システム]，[ポート] など”）を選択することにより，詳細情報表示／変更エリア，および説明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。

2. すでに本パネルがオープンしていた場合，コード生成 パネルの周辺機能ボタン“  ，  など”をクリックすることにより，詳細情報表示／変更エリア，および説明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。
3. すでに本パネルがオープンしていた場合，コード生成プレビュー パネルのファイルを選択することにより，詳細情報表示／変更エリア，および説明エリアの表示内容が該当ファイル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（[端子配置（設計ツール)]，[端子配置表]，[端子配置図]，[コード生成（設計ツール)]，周辺機能ノード“[システム]，[ポート]など”），コード生成 パネルでクリックした周辺機能ボタン“  ，  など”，コード生成プレビュー パネルで選択したファイルの種類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，コード生成 パネルでクリックした周辺機能ボタン，およびコード生成プレビュー パネルで選択したファイル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

タブを選択することにより，詳細情報を表示するカテゴリが切り替わります。

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
- [出力設定] タブ
- [マクロ設定] タブ
- [ファイル設定] タブ

[[編集] メニュー（プロパティ パネル専用部分)]

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。

[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。

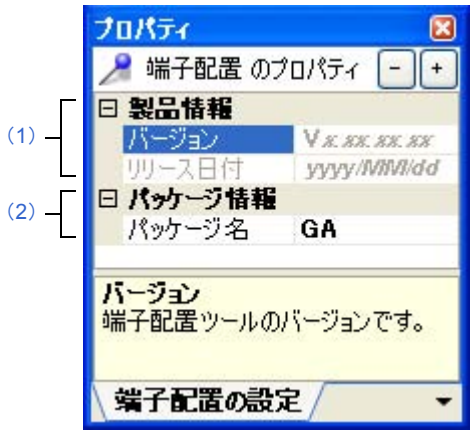
(2) 項目を編集中以外の場合

デフォルトに戻す	選択された項目をデフォルトの状態に戻します。
すべてデフォルトに戻す	すべての項目をデフォルトの状態に戻します。

［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

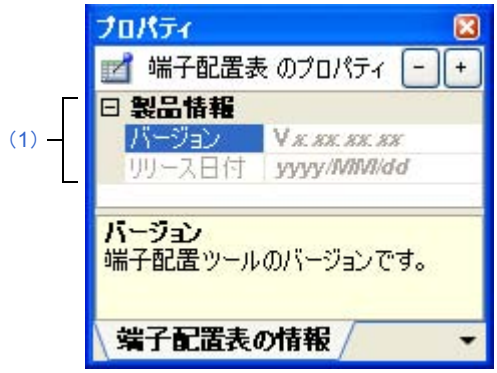
端子配置図 パネルに端子配置図として表示するマイクロコントローラの形状（パッケージ名）を選択します。

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。

備考　色設定の変更は，本エリア内のドロップダウン・リストを選択することによりオープンする以下のカラー・パレットで行います。

図 A―7　カラー・パレット

(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

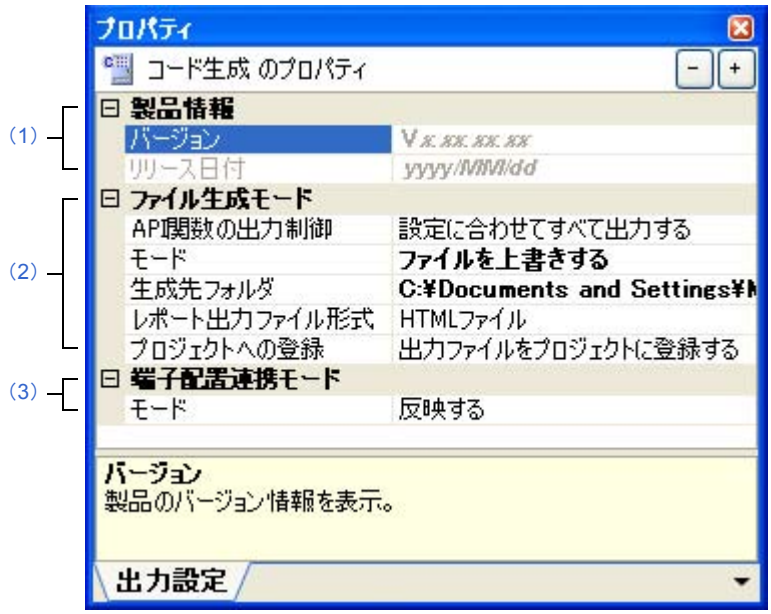
端子の付加情報を端子配置図に表示するか否かを選択します。

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。

［出力設定］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（製品情報，ファイル生成モード，端子配置連携モード）の表示，および設定の変更を行います。

図 A—8　［出力設定］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) ［製品情報］カテゴリ

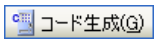
コード生成に関する製品情報（バージョン，リリース日付）の表示を行います。

バージョン	コード生成（コードライブラリ）のバージョンを表示します。
リリース日付	コード生成（コードライブラリ）のリリース日付を表示します。

(2)［ファイル生成モード］カテゴリ

コード生成のファイル生成モード（API 関数の出力制御，モードなど）の表示，および設定の変更を行います。

API 関数の出力制御	コード生成(G) ボタンをクリックした際に出力する API 関数の種類（全 API 関数，初期化用 API 関数のみ）を表示／選択します。	
	設定に合わせてすべて出力する	全 API 関数の出力となります。
	初期化関数のみ出力する	初期化用 API 関数のみの出力となります。
モード	コード生成(G) ボタンをクリックした際の動作モードを表示／選択します。 なお，［ファイル］メニュー→［コード生成レポートを保存］を選択した際の動作モードは，“ファイルを上書きする”となります。	
	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既存していた場合，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既存していた場合，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既存していた場合，該当ファイルを上書きします。
生成先フォルダ	コード生成(G) ボタンをクリックした際，［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する各種ファイル（ソース・コード，レポート・ファイル）の出力先を表示／選択します。	
レポート出力ファイル形式	［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するレポート・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）の形式を表示／選択します。	
	HTML ファイル	HTML 形式でレポート・ファイルを出力します。
	CSV ファイル	CSV 形式でレポート・ファイルを出力します。

プロジェクトへの登録	 ボタンをクリックした際に出力するソース・コードをプロジェクトに登録するか否かを選択します。	
	出力ファイルをプロジェクトに登録する	出力したソース・コードをプロジェクトに登録します。 なお，登録されたソース・コードは，プロジェクト・ツリー パネルの［ファイル］-［コード生成］ノードの直下に表示されます。
	出力ファイルをプロジェクトに登録しない	出力したソース・コードをプロジェクトに登録しません。

備考　出力先の変更は，本エリア内の［...］ボタンをクリックすることによりオープンするフォルダの参照ダイアログで行います。

(3)［端子配置連携モード］カテゴリ

コード生成と端子配置の情報連携（モード）に関する設定を行います。

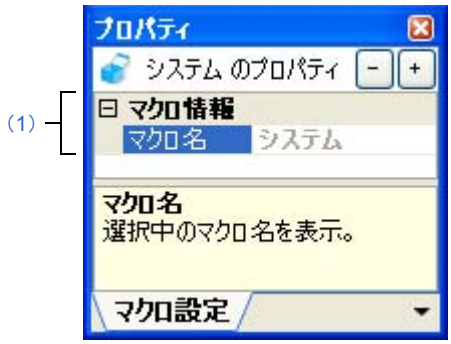
モード	 ボタンをクリックした際，コード生成 パネルで設定した各種情報を端子配置表 パネルに反映するか否かを選択します。	
	反映する	コード生成 パネルで設定した各種情報を端子配置表 パネルに反映します。
	反映しない	コード生成 パネルで設定した各種情報を端子配置表 パネルに反映しません。

備考　“反映しない”を選択した際には，  ボタンがグレー表記（非選択状態）となります。

［マクロ設定］タブ

プロジェクト・ツリー パネルで選択した周辺機能ノード“[システム]，[ポート] など”，コード生成 パネルでクリックした周辺機能ボタンの種類“ , など”に対応した情報（マクロ情報）の表示，および設定の変更を行います。

図 A―9 ［マクロ設定］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）］→周辺機能ノード“[システム]，[ポート] など”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）］→周辺機能ノード“[システム]，[ポート] など”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 パネルの周辺機能ノード“[システム]，[ポート] など”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。

2. すでに本パネルがオープンしていた場合，コード生成 パネルの周辺機能ボタン“ , など”をクリックすることにより，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) ［マクロ情報］カテゴリ

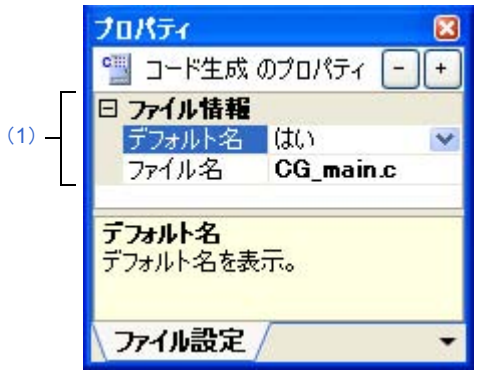
プロジェクト・ツリー パネルで選択した周辺機能ノード“[システム]，[ポート] など”，コード生成 パネルでクリックした周辺機能ボタンに関する情報（マクロ名）の表示，および設定の変更を行います。

マクロ名	プロジェクト・ツリー パネルで選択した周辺機能ノードの種類，コード生成 パネルでクリックした周辺機能ボタンの種類を表示します。

［ファイル設定］タブ

コード生成プレビュー パネルで選択したファイルの種類に対応した情報（ファイル情報）の表示，および設定の変更を行います。

図 A―10 ［ファイル設定］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］

- コード生成プレビュー パネルにおいて，ファイルを選択したのち，［表示］メニュー→［プロパティ］を選択
- コード生成プレビュー パネルにおいて，ファイルを選択したのち，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，コード生成プレビュー パネルのファイルを選択することにより，表示内容が該当ファイル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

コード生成プレビュー パネルで選択したファイルに関する情報（デフォルト名，ファイル名）の表示，および設定の変更を行います。

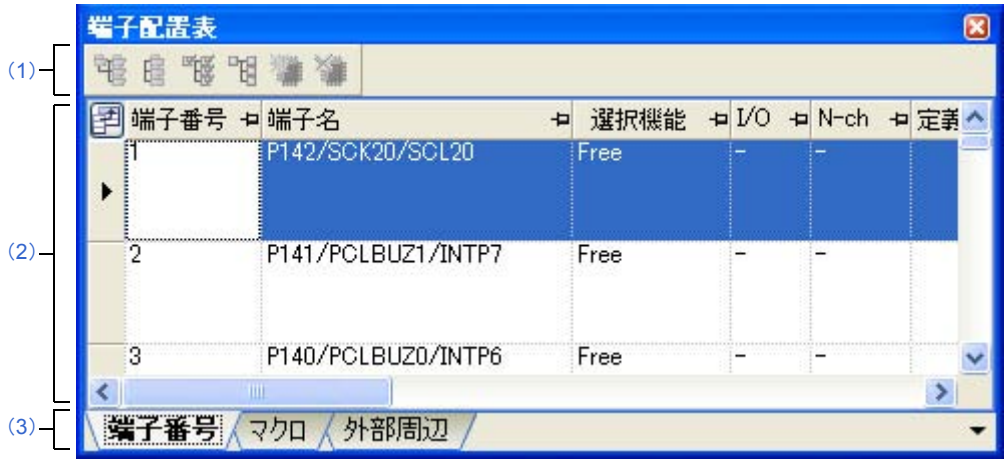
デフォルト名	コード生成プレビュー パネルで選択したファイルのファイル名がデフォルトの名前であるか否かを表示／選択します。	
	はい	該当ファイル名は，デフォルトの名前です。 本領域を“いいえ”から“はい”へと変更した際には，該当ファイル名がデフォルトの名前へと変更されます。
	いいえ	該当ファイル名は，デフォルトの名前ではありません。
ファイル名	コード生成プレビュー パネルで選択したファイルのファイル名を表示／変更します。	

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。

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—11 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］
- ［［ファイル］メニュー（端子配置表 パネル専用部分）］
- ［［ヘルプ］メニュー（端子配置表 パネル専用部分）］

［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

［ファイル］メニュー（端子配置表 パネル専用部分）

端子配置表 を保存	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保存します。
名前を付けて 端子配置表 を保存 ...	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。

［ヘルプ］メニュー（端子配置表 パネル専用部分）

端子配置表 パネルのヘルプを開く	本パネルのヘルプを表示します。

［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します。

図 A—12　［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—13 ［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。

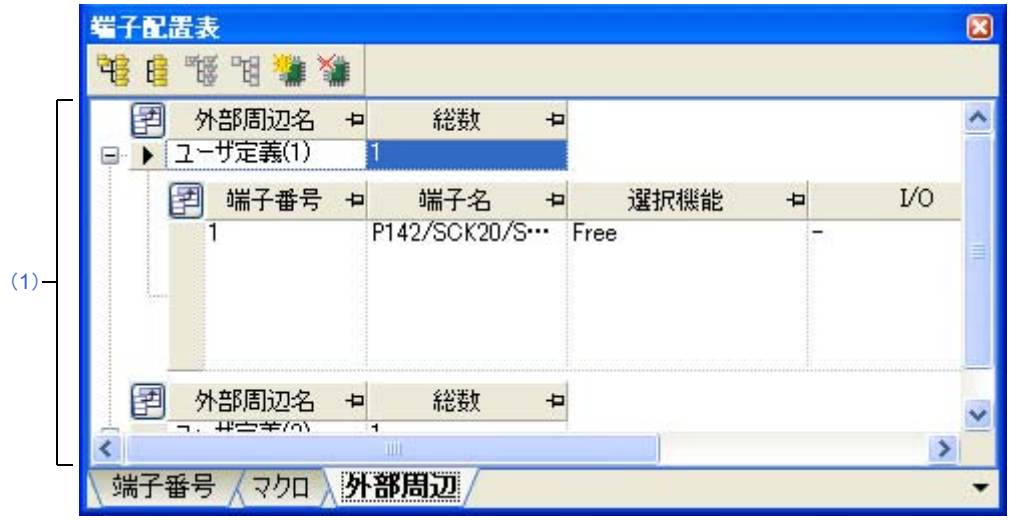
3. 列の移動（表示順序の変更）は，端子配置表の該当列をドラッグしたのち，移動先にドロップすることにより行います。

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—14 ［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。

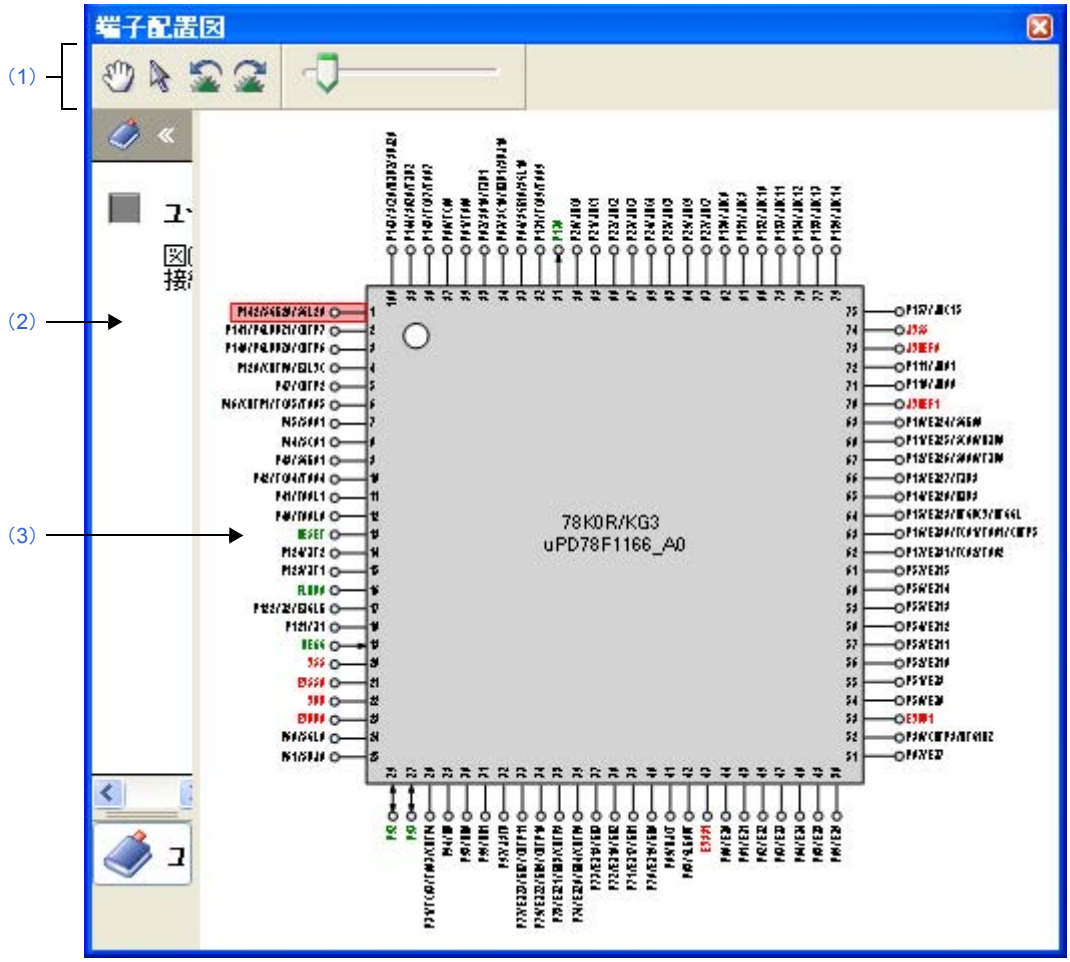
4. “ユーザ独自の列”を追加する場合，端子配置表の左上に設けられた  ボタンをクリックすることによりオープンする列の選択 ダイアログの［新しい列 ...］ボタンをクリックすることによりオープンする新しい列 ダイアログで行います。

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—15　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］
- ［［ファイル］メニュー（端子配置図 パネル専用部分）］
- ［［ヘルプ］メニュー（端子配置図 パネル専用部分）］
- ［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］タブでパッケージ名に“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。

[[ファイル] メニュー（端子配置図 パネル専用部分）]

端子配置図 を保存	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保存します。
名前を付けて 端子配置図 を保存 ...	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。

[[ヘルプ] メニュー（端子配置図 パネル専用部分）]

端子配置図 パネルのヘルプを開く	本パネルのヘルプを表示します。

[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。

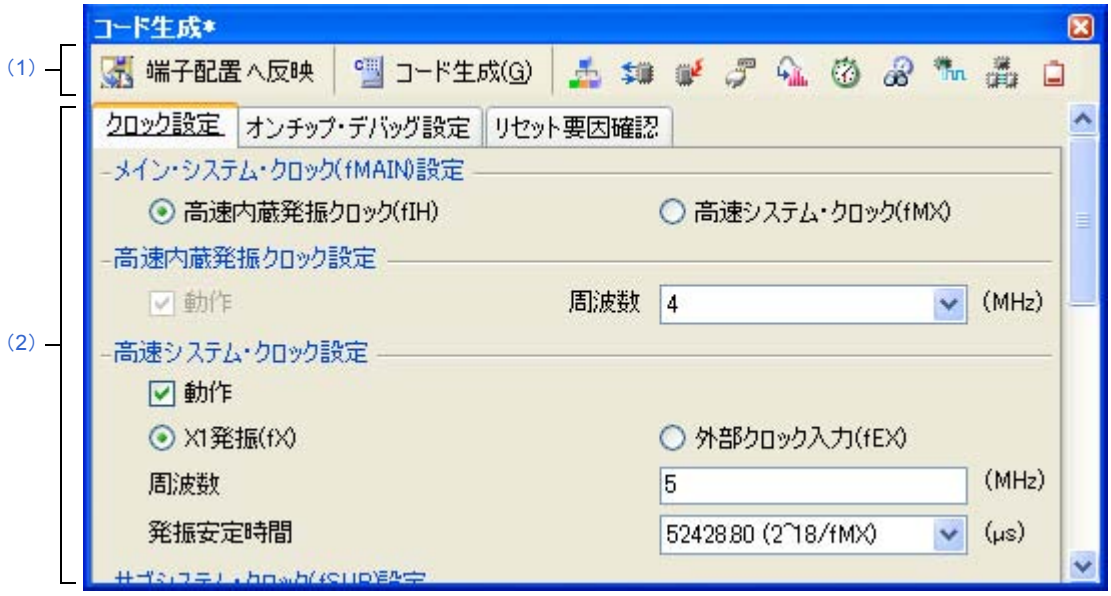
(2) 外部周辺コントローラ上で右クリックした場合

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。

コード生成 パネル

マイクロコントローラが提供している周辺機能を制御するうえで必要な情報を設定します。

図 A—16　コード生成 パネル：[システム]



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］
- ［［ファイル］メニュー（コード生成 パネル専用部分）］
- ［［ヘルプ］メニュー（コード生成 パネル専用部分）］

［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）］→周辺機能ノード“［システム］，［ポート］など”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）］→周辺機能ノード“［システム］，［ポート］など”を選択したのち，［Enter］キーを押下

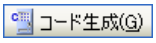
備考　すでに本パネルがオープンしていた場合，周辺機能ボタン“  ，  など”をクリックすることにより，情報設定エリアの表示内容が該当ボタンに対応したものへと切り替わります。

［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。

	本パネルで設定した各種情報を端子配置表 パネルに反映するとともに，出力パネルに変更内容を出力します。 なお，本ボタンは，［出力設定］タブの［端子配置連携モード］カテゴリで“反映しない”が選択されている際には，グレー表記（非選択状態）となります。
	プロパティ パネルの［出力設定］タブ→［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。
	情報設定エリアの表示内容を“マイクロコントローラが提供しているクロック発生回路の機能，オンチップ・デバッグ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［システム］”へと切り替えます。
	情報設定エリアの表示内容を“マイクロコントローラが提供している外部バス・インタフェースの機能（外部バスを内蔵 ROM，RAM，SFR 以外の領域に接続する機能）を制御するうえで必要な情報を設定するための［外部バス］”へと切り替えます。
	情報設定エリアの表示内容を“マイクロコントローラが提供しているポートの機能を制御するうえで必要な情報を設定するための［ポート］”へと切り替えます。
	情報設定エリアの表示内容を“マイクロコントローラが提供している割り込み／キー割り込みの機能を制御するうえで必要な情報を設定するための［割り込み］”へと切り替えます。
	情報設定エリアの表示内容を“マイクロコントローラが提供しているシリアル・アレイ・ユニット，およびシリアル・インタフェースの機能を制御するうえで必要な情報を設定するための［シリアル］”へと切り替えます。
	情報設定エリアの表示内容を“マイクロコントローラが提供しているオペアンプの機能を制御するうえで必要な情報を設定するための［オペアンプ］”へと切り替えます。
	情報設定エリアの表示内容を“マイクロコントローラが提供している低電圧検出回路の機能を制御するうえで必要な情報を設定するための［コンパレータ／PG アンプ］”へと切り替えます。
	情報設定エリアの表示内容を“マイクロコントローラが提供している A/D コンバータの機能を制御するうえで必要な情報を設定するための［A/D コンバータ］”へと切り替えます。
	情報設定エリアの表示内容を“マイクロコントローラが提供している D/A コンバータの機能を制御するうえで必要な情報を設定するための［D/A コンバータ］”へと切り替えます。
	情報設定エリアの表示内容を“マイクロコントローラが提供しているタイマ・アレイ・ユニットの機能を制御するうえで必要な情報を設定するための［タイマ］”へと切り替えます。
	情報設定エリアの表示内容を“マイクロコントローラが提供しているウォッチドッグ・タイマの機能を制御するうえで必要な情報を設定するための［ウォッチドッグ・タイマ］”へと切り替えます。

	情報設定エリアの表示内容を“マイクロコントローラが提供しているリアルタイム・カウンタの機能を制御するうえで必要な情報を設定するための［リアルタイム・カウンタ］”へと切り替えます。
	情報設定エリアの表示内容を“マイクロコントローラが提供しているクロック出力制御回路の機能を制御するうえで必要な情報を設定するための［クロック出力］”へと切り替えます。
	情報設定エリアの表示内容を“マイクロコントローラが提供しているクロック出力／ブザー出力制御回路の機能を制御するうえで必要な情報を設定するための［クロック出力／ブザー出力］”へと切り替えます。
	情報設定エリアの表示内容を“マイクロコントローラが提供している LCD コントローラ／ドライバの機能を制御するうえで必要な情報を設定するための［LCD コントローラ／ドライバ］”へと切り替えます。
	情報設定エリアの表示内容を“マイクロコントローラが提供している DMA コントローラの機能を制御するうえで必要な情報を設定するための［DMA コントローラ］”へと切り替えます。
	情報設定エリアの表示内容を“マイクロコントローラが提供している低電圧検出回路の機能を制御するうえで必要な情報を設定するための［低電圧検出回路］”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。

［［ファイル］メニュー（コード生成 パネル専用部分）］

コード生成レポート を保存	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。

備考 1. レポート・ファイルの出力形式はプロパティ パネルの［出力設定］タブ→［レポート出力ファイル形式］で選択された形式（HTML 形式，または CSV 形式）となります。

2. レポート・ファイルの出力先は，プロパティ パネルの［出力設定］タブ→［生成先フォルダ］で指定されたフォルダとなります。

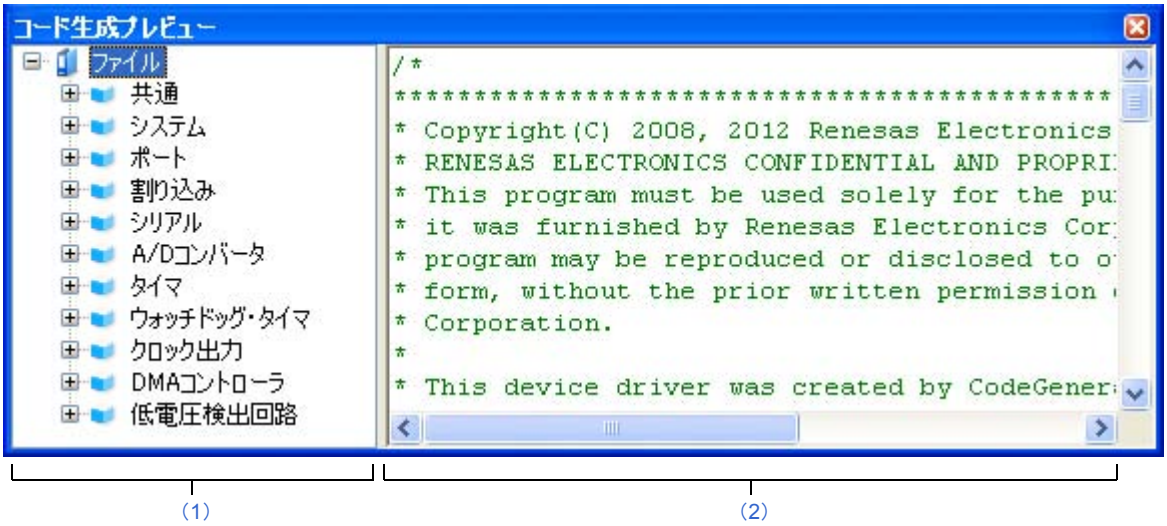
［［ヘルプ］メニュー（コード生成 パネル専用部分）］

コード生成 パネルのヘルプを開く	本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

コード生成 パネルの コード生成(G) ボタンをクリックした際に出力されるソース・コード（デバイス・ドライバ・プログラム）の出力有無を API 関数単位で確認／設定するとともに，コード生成 パネルで設定した情報に応じたソース・コードの確認を行います。

図 A―17 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］
- ［［ファイル］メニュー（コード生成プレビュー パネル専用部分）］
- ［［ヘルプ］メニュー（コード生成プレビュー パネル専用部分）］
- ［コンテキスト・メニュー］

［オープン方法］

- ［表示］メニュー→［コード生成プレビュー］を選択

［各エリアの説明］

(1) プレビュー・ツリー

コード生成 パネルの コード生成(G) ボタンをクリックした際に出力されるソース・コード（デバイス・ドライバ・プログラム）の出力有無を API 関数単位で確認／設定します。

備考 1. 本ツリー内のソース・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. 出力有無の設定は，ツリー内のアイコン上にマウス・カーソルを移動した際，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（コードを生成する，コードを生成しない）から行います。
3. 出力有無の設定状況については，アイコン種別により確認することができます。

表 A―2　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・コードは，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・コードは，出力されます。
	該当 API 関数のソース・コードは，出力されません。

(2) ソース・コード表示エリア

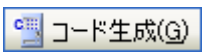
コード生成 パネルで設定した情報に応じたソース・コード（デバイス・ドライバ・プログラム）の確認を行います。

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―3　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（シリアル・アレイ・ユニット用 API 関数など）については，ソース・コードの出力時（コード生成 パネルの 

ボタンをクリックした際）にレジスタ値 SFR 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. プレビュー・ツリー内のソース・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。

[[ファイル] メニュー（コード生成プレビュー パネル専用部分)]

コード生成レポート を保存	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。

備考 1. レポート・ファイルの出力形式はプロパティ パネルの［出力設定］タブ→［レポート出力ファイル形式］で選択された形式（HTML 形式，または CSV 形式）となります。

2. レポート・ファイルの出力先は，プロパティ パネルの［出力設定］タブ→［生成先フォルダ］で指定されたフォルダとなります。

［[ヘルプ］メニュー（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 パネルのヘルプを開く	本パネルのヘルプを表示します。

［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。

コードを生成する	選択された API 関数のソース・コードをプロパティ パネルの［出力設定］タブ→［生成先フォルダ］で指定されたフォルダに出力するための設定を行います。 なお，本コンテキスト・メニューをクリックすることにより，該当 API 関数のアイコンは，  から  へと変化します。 本項目は，選択された API 関数が初期化用 API 関数以外であり，プロパティ パネルの［出力設定］タブ→［API 関数の出力制御］で“初期化関数のみ出力する”が選択されている際には，グレー表記（非選択状態）となります。
コードを生成しない	選択された API 関数のソース・コードをコード生成 パネルの  ボタンがクリックされた際に出力しないための設定を行います。 なお，本コンテキスト・メニューをクリックすることにより，該当 API 関数のアイコンは，  から  へと変化します。
名前を変更する	選択されたファイル名／ API 関数名の部位が該当名称を編集するためにエディット・ボックス化されます。 該当エディット・ボックスの文字列を編集することにより，ファイル名／ API 関数名が変更されます。
名前を元に戻す	選択されたファイル名／ API 関数名を編集前の状態に戻します。
プロパティ	選択されたファイルに対応した情報を保持したプロパティ パネルをオープンします。

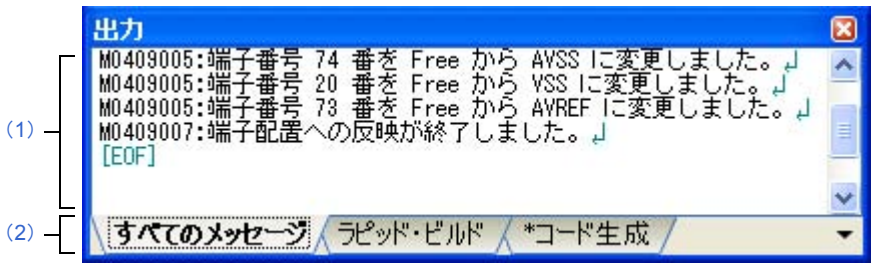
出力 パネル

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表示を行います。

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—18　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］
- ［［ファイル］メニュー（出力 パネル専用部分）］
- ［［編集］メニュー（出力 パネル専用部分）］
- ［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，メッセージの表示色は，出力メッセージの種別により，次のように異なります（表示の際の文字色／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）。

メッセージ種別	表示例（デフォルト）			説明
通常メッセージ	Aaあぁアァ亜宇	文字色	黒	何らかの情報を通知する際に表示されます。
		背景色	白	

メッセージ種別	表示例（デフォルト）			説明
警告メッセージ	Aaあぁアァ亜宇	文字色	青	操作に対して，何らかの警告を通知する際に表示されます。
		背景色	標準色	
エラー・メッセージ	Aaあぁアァ亜宇	文字色	赤	致命的なエラー，または操作ミスにより実行が不可能な場合に表示されます。
		背景色	グレー	

(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。

[[ファイル] メニュー（出力 パネル専用部分）]

出力 - *タブ名* を保存	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 ...	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。

[[編集] メニュー（出力 パネル専用部分）]

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。
検索 ...	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 ...	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。

[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。
検索の中止	実行中の文字列検索を中止します。 文字列検索を非実行中の場合，本項目は無効となります。
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追加／削除を行います。

図 A—19 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。

図 A—20　操作対象（［端子番号］タブ）

図 A—21　操作対象（［マクロ］タブ：第 1 階層）

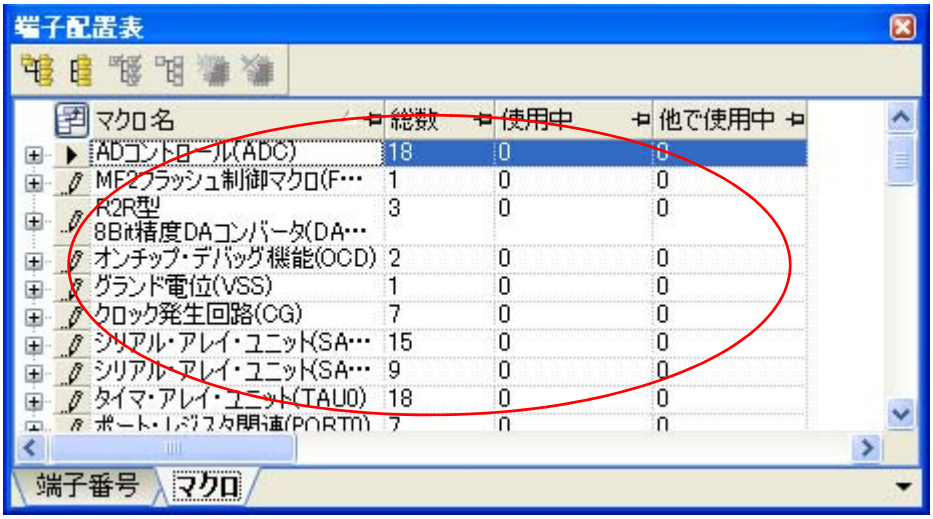


図 A—22 操作対象（[マクロ] タブ：第 2 階層）

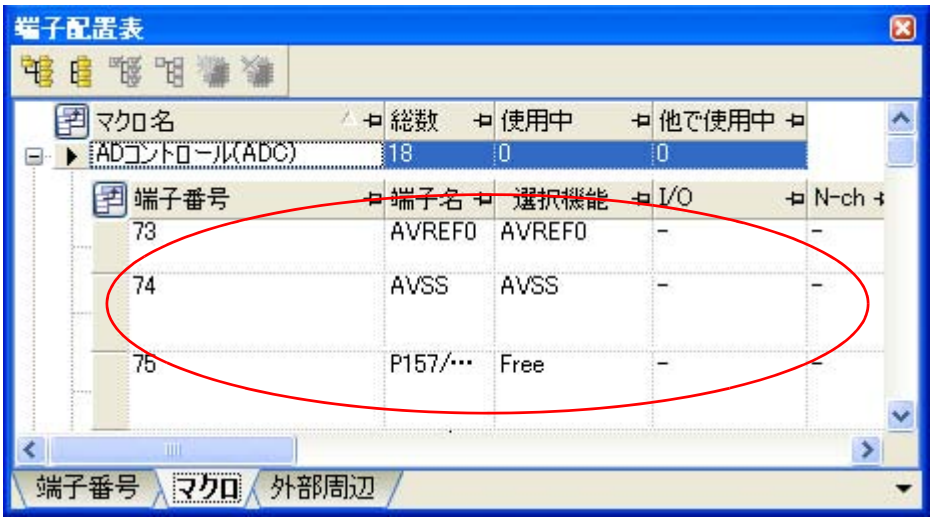


図 A—23 操作対象（[外部周辺] タブ：第 1 階層）

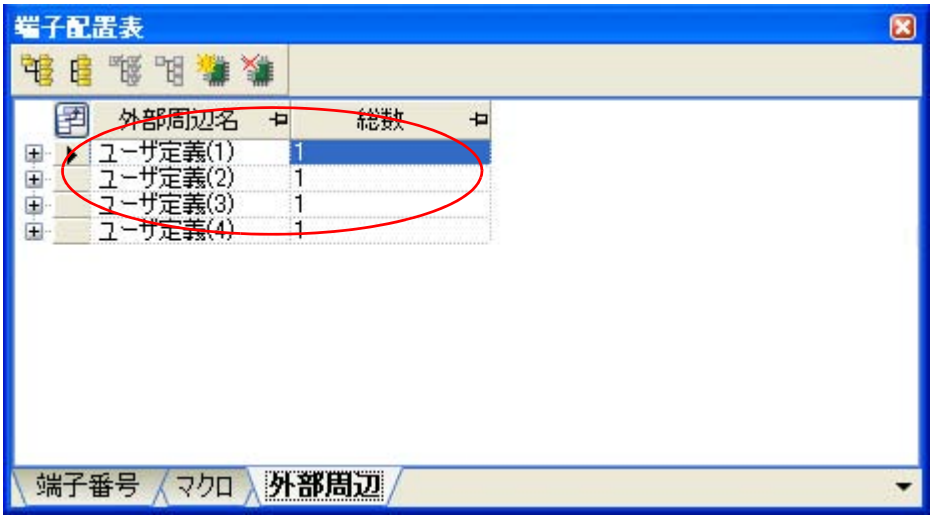
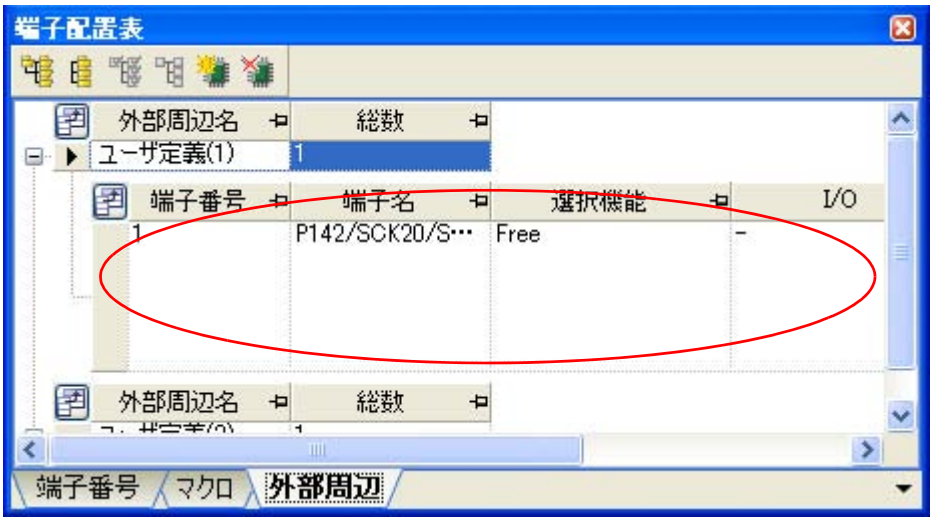


図 A―24　操作対象（[外部周辺] タブ：第 2 階層）



(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—25　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。

フォルダの参照 ダイアログ

ファイル（ソース・コード，レポート・ファイルなど）の出力先を設定します。

図 A—26 フォルダの参照 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］

- プロパティ パネルの［出力設定］タブにおいて，［生成先フォルダ］の［...］ボタンをクリック

［各エリアの説明］

（1）保存する場所エリア

ファイル（ソース・コード，レポート・ファイルなど）を出力するフォルダを選択します。

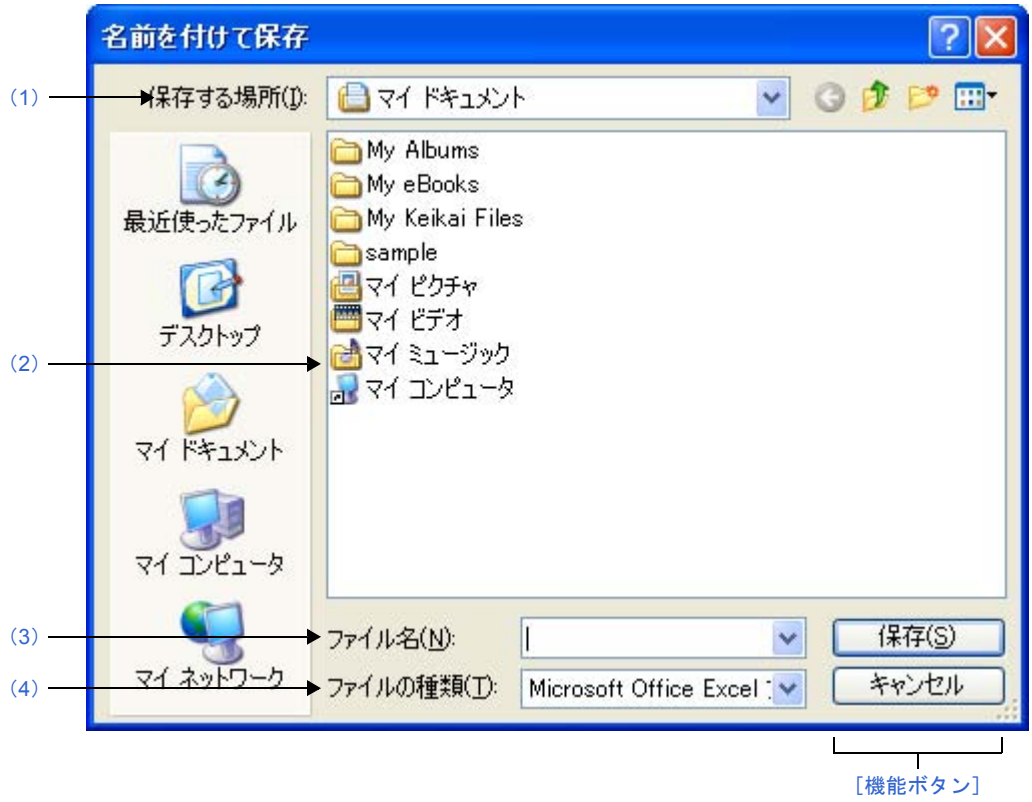
［機能ボタン］

ボタン	機能
新しいフォルダの作成	保存する場所エリアで選択されたフォルダの直下に“新しいフォルダ”を新規に作成します。
OK	ファイルの出力先を保存する場所エリアで選択されたフォルダに設定します。
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（レポート・ファイルなど）に名前を付けて保存します。

図 A—27　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→［名前を付けて *＜対象＞* を保存］を選択

［各エリアの説明］

(1) ［保存する場所］

ファイル（レポート・ファイルなど）を出力するフォルダを選択します。

(2) ファイルの一覧エリア

［保存する場所］，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］

出力するファイルのファイル名を指定します。

(4)［ファイルの種類］

出力するファイルの種類（ファイル・タイプ）を選択します。

Microsoft Office Excel ブック（*.xls）	Microsoft Office Excel ブック形式
ビットマップ（*.bmp）	ビット・マップ形式
PNG（*.png）	PNG 形式
JPEG（*.jpg）	JPEG 形式
EMF（*.emf）	EMF 形式

［機能ボタン］

ボタン	機能
保存	［保存する場所］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　概　　要

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

出力単位	ファイル名	出力内容
各周辺機能	CG_ *周辺機能名* .c	初期化関数，API 関数
	CG_ *周辺機能名* _user.c	割り込み関数（MD_INT*xxx*），コールバック関数
	CG_ *周辺機能名* .h	レジスタへの代入値マクロを定義
プロジェクト	CG_main.c	main 関数，R_MAIN_UserInit 関数
	CG_systeminit.c	各周辺機能の初期化関数コール CG_ReadResetSource のコール
	CG_macrodriver.h	全ソース・ファイルで共通使用するマクロを定義
	CG_userdefine.h	空ファイル（ユーザ定義用）
	CG_lk.dr	リンク・ディレクティブ

B. 2　出力ファイル

以下に，コード生成が出力するファイル（各周辺機能）を示します。

表 B—2　出力ファイル（各周辺機能）

周辺機能	ファイル名	含まれる API 関数名
システム	CG_system.c	CLOCK_Init CG_ChangeClockMode CG_ChangeFrequency CG_SelectPowerSaveMode CG_SelectStabTime
	CG_system_user.c	CLOCK_UserInit CG_ReadResetSource
	CG_system.h	—
外部バス	CG_bus.c	BUS_Init BUS_PowerOff
	CG_bus_user.c	BUS_UserInit

周辺機能	ファイル名	含まれる API 関数名
外部バス	CG_bus.h	—
ポート	CG_port.c	PORT_Init PORT_ChangePmnInput PORT_ChangePmnOutput
	CG_port_user.c	PORT_UserInit
	CG_port.h	—
割り込み	CG_int.c	INTP_Init KEY_Init INT_MaskableInterruptEnable INTPn_Disable INTPn_Enable KEY_Disable KEY_Enable
	CG_int_user.c	INTP_UserInit KEY_UserInit MD_INTP*n* MD_INTKR
	CG_int.h	—
シリアル	CG_serial.c	SAUm_Init SAUm_PowerOff UARTn_Init UARTn_Start UARTn_Stop UARTn_SendData UARTn_ReceiveData CSImn_Init CSImn_Start CSImn_Stop CSImn_SendData CSImn_ReceiveData CSImn_SendReceiveData IICmn_Init IICmn_Stop IICmn_MasterSendStart IICmn_MasterReceiveStart IICmn_StartCondition IICmn_StopCondition UARTFn_Init UARTFn_PowerOff UARTFn_Start UARTFn_Stop UARTFn_SendData UARTFn_ReceiveData

周辺機能	ファイル名	含まれる API 関数名
シリアル	CG_serial.c	UARTFn_SetComparisonData UARTFn_DataComparisonEnable UARTFn_DataComparisonDisable IICA_Init IICA_PowerOff IICA_Stop IICA_MasterSendStart IICA_MasterReceiveStart IICA_StopCondition IICA_SlaveSendStart IICA_SlaveReceiveStart IICn_Init IICn_Stop IICn_MasterSendStart IICn_MasterReceiveStart IICn_SlaveSendStart IICn_SlaveReceiveStart
	CG_serial_user.c	SAUm_UserInit UARTn_SendEndCallback UARTn_ReceiveEndCallback UARTn_SoftOverRunCallback UARTn_ErrorCallback CSImn_SendEndCallback CSImn_ReceiveEndCallback CSImn_ErrorCallback IICmn_MasterSendEndCallback IICmn_MasterReceiveEndCallback IICmn_MasterErrorCallback UARTFn_SendEndCallback UARTFn_ReceiveEndCallback UARTFn_SoftOverRunCallback UARTFn_ExpBitCetectCallback UARTFn_IDMatchCallback UARTFn_ErrorCallback IICA_UserInit IICA_MasterSendEndCallback IICA_MasterReceiveEndCallback IICA_MasterErrorCallback IICA_SlaveSendEndCallback IICA_SlaveReceiveEndCallback IICA_SlaveErrorCallback IICA_GetStopConditionCallback IICn_UserInit IICn_MasterSendEndCallback IICn_MasterReceiveEndCallback

周辺機能	ファイル名	含まれる API 関数名
シリアル	CG_serial_user.c	IICn_MasterErrorCallback IICn_SlaveSendEndCallback IICn_SlaveReceiveEndCallback IICn_SlaveErrorCallback IICn_GetStopConditionCallback MD_INTSR*n* MD_INTSRE*n* MD_INTST*n* MD_INTCSI*mn* MD_INTIIC*mn* MD_INTLT*n* MD_INTLR*n* MD_INTLS*n* MD_INTIIC*n* MD_INTIICA
	CG_serial.h	—
オペアンプ	CG_opamp.c	OPAMP_Init AMPn_Start AMPn_Stop
	CG_opamp_user.c	OPAMP_UserInit
	CG_opamp.h	—
コンパレータ／PG アンプ	CG_cmppga.c	CMPPGA_Init CMPPGA_PowerOff CMPPGA_Start CMPPGA_Stop CMPPGA_ChangeCMPnRefVoltage CMPPGA_ChangePGAFactor
	CG_cmppga_user.c	CMPPGA_UserInit MD_INTCMP*n*
	CG_cmppga.h	—
A/D コンバータ	CG_ad.c	AD_Init AD_PowerOff AD_ComparatorOn AD_ComparatorOff AD_Start AD_Stop AD_SelectADChannel AD_Read AD_ReadByte
	CG_ad_user.c	AD_UserInit MD_INTAD
	CG_ad.h	—

周辺機能	ファイル名	含まれる API 関数名
D/A コンバータ	CG_da.c	DA_Init DA_PowerOff DAn_Start DAn_Stop DAn_SetValue DAn_Set8BitsValue DAn_Set12BitsValue
	CG_da_user.c	DA_UserInit
	CG_ds.h	—
タイマ	CG_timer.c	TAUm_Init TAUm_PowerOff TAUm_ChannelN_Start TAUm_ChannelN_Stop TAUm_ChannelN_ChangeCondition TAUm_ChannelN_ChangeTimerCondition TAUm_ChannelN_GetPulseWidth TAUm_ChannelN_ChangeDuty TAUm_ChannelN_SoftWareTriggerOn
	CG_timer_user.c	TAUm_UserInit MD_INTTM*mn*
	CG_timer.h	—
ウォッチドッグ・タイマ	CG_wdt.c	WDT_Init WDT_Restart
	CG_wdt_user.c	WDT_UserInit
	CG_wdt.h	—
リアルタイム・カウンタ	CG_rtc.c	RTC_Init RTC_PowerOff RTC_CounterEnable RTC_CounterDisable RTC_SetHourSystem RTC_CounterSet RTC_CounterGet RTC_ConstPeriodInterruptEnable RTC_ConstPeriodInterruptDisable RTC_AlarmEnable RTC_AlarmDisable RTC_AlarmSet RTC_AlarmGet RTC_IntervalStart RTC_IntervalStop RTC_IntervalInterruptEnable RTC_IntervalInterruptDisable RTC_RTC1HZ_OutputEnable

周辺機能	ファイル名	含まれる API 関数名
リアルタイム・カウンタ	CG_rtc.c	RTC_RTC1HZ_OutputDisable RTC_RTCCL_OutputEnable RTC_RTCCL_OutputDisable RTC_RTCCL_OutputDisable RTC_RTCDIV_OutputEnable RTC_RTCDIV_OutputDisable RTC_ChangeCorrectionValue
	CG_rtc_user.c	RTC_UserInit RTC_ConstPeriodInterruptCallback RTC_AlarmInterruptCallback MD_INTRTC MD_INTRTCI
	CG_rtc.h	—
クロック出力	CG_pcl.c	PCL_Init PCL_Start PCL_Stop PCL_ChangeFreq
	CG_pcl_user.c	PCL_UserInit
	CG_pcl.h	—
クロック出力／ブザー出力	CG_pclbuz.c	PCLBUZn_Init PCLBUZn_Start PCLBUZn_Stop PCLBUZn_ChangeFreq
	CG_pclbuz_user.c	PCLBUZn_UserInit
	CG_pclbuz.h	—
LCD コントローラ／ドライバ	CG_lcd.c	LCD_Init LCD_DisplayOn LCD_DisplayOff LCD_VoltageOn LCD_VoltageOff
	CG_lcd_user.c	LCD_UserInit
	CG_lcd.h	—
DMA コントローラ	CG_dma.c	DMAn_Init DMAn_Enable DMAn_Disable DMAn_Hold DMAn_Restart DMAn_CheckStatus DMAn_SetData DMAn_SoftwareTriggerOn
	CG_dma_user.c	DMAn_UserInit MD_INTDMA*n*

周辺機能	ファイル名	含まれる API 関数名
DMA コントローラ	CG_dma.h	—
低電圧検出回路	CG_lvi.c	LVI_Init LVI_InterruptModeStart LVI_ResetModeStart LVI_Stop LVI_SetLVILevel
	CG_lvi_user.c	LVI_UserInit MD_INTLVI
	CG_lvi.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　出力関数

以下に，コード生成が出力する API 関数の一覧を示します。

表 C—1　API 関数一覧

周辺機能	API 関数名	機能概要
システム	CLOCK_Init	クロック発生回路の機能，オンチップ・デバッグ機能などを制御するうえで必要となる初期化処理を行います。
	CLOCK_UserInit	クロック発生回路，オンチップ・デバッグなどに関するユーザ独自の初期化処理を行います。
	CG_ReadResetSource	内部リセットの発生に伴う処理を行います。
	CG_ChangeClockMode	CPU クロック／周辺ハードウエア・クロックを変更します。
	CG_ChangeFrequency	CPU クロック／周辺ハードウエア・クロックの分周比を変更します。

周辺機能	API 関数名	機能概要
システム	CG_SelectPowerSaveMode	CPU のスタンバイ・モードを設定します。
	CG_SelectStabTime	X1 クロックの発振安定時間を設定します。
外部バス	BUS_Init	外部バス・インタフェースの機能（外部バスを内蔵 ROM，RAM，SFR 以外の領域に接続する機能）を制御するうえで必要となる初期化処理を行います。
	BUS_UserInit	外部バス・インタフェースに関するユーザ独自の初期化処理を行います。
	BUS_PowerOff	外部バス・インタフェースに対するクロック供給を停止します。
ポート	PORT_Init	ポートの機能を制御するうえで必要となる初期化処理を行います。
	PORT_UserInit	ポートに関するユーザ独自の初期化処理を行います。
	PORT_ChangePmnInput	端子の入出力モードを出力モードから入力モードへと切り替えます。
	PORT_ChangePmnOutput	端子の入出力モードを入力モードから出力モードへと切り替えます。
割り込み	INTP_Init	外部割り込み INTP*n* の機能を制御するうえで必要となる初期化処理を行います。
	INTP_UserInit	外部割り込み INTP*n* に関するユーザ独自の初期化処理を行います。
	KEY_Init	キー割り込み INTKR の機能を制御するうえで必要となる初期化処理を行います。
	KEY_UserInit	キー割り込み INTKR に関するユーザ独自の初期化処理を行います。
	INT_MaskableInterruptEnable	マスカブル割り込みの受け付けを禁止／許可します。
	INTPn_Disable	マスカブル割り込み（外部割込み要求）INTP*n* の受け付けを禁止します。
	INTPn_Enable	マスカブル割り込み（外部割込み要求）INTP*n* の受け付けを許可します。
	KEY_Disable	キー割り込み INTKR の受け付けを禁止します。
	KEY_Enable	キー割り込み INTKR の受け付けを許可します。
シリアル	SAUm_Init	シリアル・アレイ・ユニット，およびシリアル・インタフェースの機能を制御するうえで必要となる初期化処理を行います。
	SAUm_UserInit	シリアル・アレイ・ユニット，およびシリアル・インタフェースに関するユーザ独自の初期化処理を行います。
	SAUm_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。

周辺機能	API 関数名	機能概要
シリアル	UARTn_Init	シリアル・インタフェース（UART）用チャネルの初期化処理を行います。
	UARTn_Start	UART 通信を待機状態にします。
	UARTn_Stop	UART 通信を終了します。
	UARTn_SendData	データの UART 送信を開始します。
	UARTn_ReceiveData	データの UART 受信を開始します。
	UARTn_SendEndCallback	UART 送信完了割り込み INTST*n* の発生に伴う処理を行います。
	UARTn_ReceiveEndCallback	UART 受信完了割り込み INTSR*n* の発生に伴う処理を行います。
	UARTn_SoftOverRunCallback	UART 受信完了割り込み INTSR*n* の発生に伴う処理を行います。
	UARTn_ErrorCallback	UART 通信におけるエラー割り込み INTSRE*n* の発生に伴う処理を行います。
	CSImn_Init	シリアル・インタフェース（CSI）用チャネルの初期化処理を行います。
	CSImn_Start	CSI 通信を待機状態にします。
	CSImn_Stop	CSI 通信を終了します。
	CSImn_SendData	データの CSI 送信を開始します。
	CSImn_ReceiveData	データの CSI 受信を開始します。
	CSImn_SendReceiveData	データの CSI 送受信を開始します。
	CSImn_SendEndCallback	CSI 通信完了割り込み INTCSI*mn* の発生に伴う処理を行います。
	CSImn_ReceiveEndCallback	CSI 通信完了割り込み INTCSI*mn* の発生に伴う処理を行います。
	CSImn_ErrorCallback	CSI 通信におけるエラー割り込み INTSRE*n* の発生に伴う処理を行います。
	IICmn_Init	シリアル・インタフェース（簡易 IIC）用チャネルの初期化処理を行います。
	IICmn_Stop	簡易 IIC 通信を終了します。
	IICmn_MasterSendStart	簡易 IIC マスタ送信を開始します。
	IICmn_MasterReceiveStart	簡易 IIC マスタ受信を開始します。
	IICmn_StartCondition	スタート・コンディションを発生します。
	IICmn_StopCondition	ストップ・コンディションを発生します。
	IICmn_MasterSendEndCallback	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。
	IICmn_MasterReceiveEndCallback	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。

周辺機能	API 関数名	機能概要
シリアル	IICmn_MasterErrorCallback	簡易 IIC 通信におけるパリティ・エラー（ACK エラー）の検出に伴う処理を行います。
	UARTFn_Init	シリアル・インタフェース（UARTF*n*）の初期化処理を行います。
	UARTFn_PowerOff	シリアル・インタフェース（UARTF*n*）に対するクロック供給を停止します。
	UARTFn_Start	UARTF 通信を待機状態にします。
	UARTFn_Stop	UARTF 通信を終了します。
	UARTFn_SendData	データの UARTF 送信を開始します。
	UARTFn_ReceiveData	データの UARTF 受信を終了します。
	UARTFn_SetComparisonData	受信データと比較するデータを設定します。
	UARTFn_DataComparisonEnable	データの比較を開始します。
	UARTFn_DataComparisonDisable	データの比較を終了します。
	UARTFn_SendEndCallback	送信割り込み INTLT*n* の発生に伴う処理を行います。
	UARTFn_ReceiveEndCallback	受信完了割り込み INTLR*n* の発生に伴う処理を行います。
	UARTFn_SoftOverRunCallback	受信完了割り込み INTLR*n* の発生に伴う処理を行います。
	UARTFn_ExpBitCetectCallback	ステータス割り込み INTLS*n* の発生に伴う処理を行います。
	UARTFn_IDMatchCallback	ステータス割り込み INTLS*n* の発生に伴う処理を行います。
	UARTFn_ErrorCallback	ステータス割り込み INTLS*n* の発生に伴う処理を行います。
	IICA_Init	シリアル・インタフェース（IICA）の初期化処理を行います。
	IICA_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。
	IICA_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。
	IICA_Stop	IICA 通信を終了します。
	IICA_MasterSendStart	IICA マスタ送信を開始します。
	IICA_MasterReceiveStart	IICA マスタ受信を開始します。
	IICA_StopCondition	ストップ・コンディションを発生します。
	IICA_MasterSendEndCallback	IICA 通信完了割り込み INTIICA の発生に伴う処理を行います。
	IICA_MasterReceiveEndCallback	IICA 通信完了割り込み INTIICA の発生に伴う処理を行います。
	IICA_MasterErrorCallback	IICA マスタ通信におけるエラーの検出に伴う処理を行います。

周辺機能	API 関数名	機能概要
シリアル	IICA_SlaveSendStart	IICA スレーブ送信を開始します。
	IICA_SlaveReceiveStart	IICA スレーブ受信を開始します。
	IICA_SlaveSendEndCallback	IICA 通信完了割り込み INTIICA の発生に伴う処理を行います。
	IICA_SlaveReceiveEndCallback	IICA 通信完了割り込み INTIICA の発生に伴う処理を行います。
	IICA_SlaveErrorCallback	IICA スレーブ通信におけるエラーの検出に伴う処理を行います。
	IICA_GetStopConditionCallback	IICA スレーブ通信におけるストップ・コンディションの検出に伴う処理を行います。
	IICn_Init	シリアル・インタフェース（IIC*n*）の初期化処理を行います。
	IICn_UserInit	シリアル・インタフェース（IIC*n*）に関するユーザ独自の初期化処理を行います。
	IICn_Stop	IIC*n* 通信を終了します。
	IICn_MasterSendStart	IIC*n* マスタ送信を開始します。
	IICn_MasterReceiveStart	IIC*n* マスタ受信を開始します。
	IICn_MasterSendEndCallback	IIC*n* 通信完了割り込み INTIIC*n* の発生に伴う処理を行います。
	IICn_MasterReceiveEndCallback	IIC*n* 通信完了割り込み INTIIC*n* の発生に伴う処理を行います。
	IICn_MasterErrorCallback	IIC*n* マスタ通信におけるエラーの検出に伴う処理を行います。
	IICn_SlaveSendStart	IIC*n* スレーブ送信を開始します。
	IICn_SlaveReceiveStart	IIC*n* スレーブ受信を開始します。
	IICn_SlaveSendEndCallback	IIC*n* 通信完了割り込み INTIIC*n* の発生に伴う処理を行います。
	IICn_SlaveReceiveEndCallback	IIC*n* 通信完了割り込み INTIIC*n* の発生に伴う処理を行います。
	IICn_SlaveErrorCallback	IIC*n* スレーブ通信におけるエラーの検出に伴う処理を行います。
	IICn_GetStopConditionCallback	IIC*n* スレーブ通信におけるストップ・コンディションの検出に伴う処理を行います。
オペアンプ	OPAMP_Init	オペアンプの機能を制御するうえで必要となる初期化処理を行います。
	OPAMP_UserInit	オペアンプに関するユーザ独自の初期化処理を行います。
	AMPn_Start	オペアンプ *n*（シングル・アンプ・モード）の動作を開始します。

周辺機能	API 関数名	機能概要
オペアンプ	AMPn_Stop	オペアンプ *n*（シングル・アンプ・モード）の動作を停止します。
コンパレータ／PG アンプ	CMPPGA_Init	コンパレータ／プログラマブル・ゲイン・アンプの機能を制御するうえで必要となる初期化処理を行います。
	CMPPGA_UserInit	コンパレータ／プログラマブル・ゲイン・アンプに関するユーザ独自の初期化処理を行います。
	CMPPGA_PowerOff	コンパレータ／プログラマブル・ゲイン・アンプに対するクロック供給を停止します。
	CMPPGA_Start	コンパレータ／プログラマブル・ゲイン・アンプの動作を開始します。
	CMPPGA_Stop	コンパレータ／プログラマブル・ゲイン・アンプの動作を停止します。
	CMPPGA_ChangeCMPnRefVoltage	コンパレータ *n* の内蔵基準電圧を設定します。
	CMPPGA_ChangePGAFactor	プログラマブル・ゲイン・アンプにおける入力電圧の増幅率を設定します。
A/D コンバータ	AD_Init	A/D コンバータの機能を制御するうえで必要となる初期化処理を行います。
	AD_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。
	AD_PowerOff	A/D コンバータに対するクロック供給を停止します。
	AD_ComparatorOn	電圧コンパレータを動作許可状態に設定します。
	AD_ComparatorOff	電圧コンパレータを動作停止状態に設定します。
	AD_Start	A/D 変換を開始します。
	AD_Stop	A/D 変換を終了します。
	AD_SelectADChannel	A/D 変換するアナログ電圧の入力端子を設定します。
	AD_Read	A/D 変換結果を読み出します。
	AD_ReadByte	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。
D/A コンバータ	DA_Init	D/A コンバータの機能を制御するうえで必要となる初期化処理を行います。
	DA_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。
	DA_PowerOff	D/A コンバータに対するクロック供給を停止します。
	DAn_Start	D/A 変換を開始します。
	DAn_Stop	D/A 変換を終了します。
	DAn_SetValue	ANO*n* 端子に出力するアナログ電圧値を設定します。
	DAn_Set8BitsValue	ANO*n* 端子に出力するアナログ電圧値（8 ビット）を設定します。

周辺機能	API 関数名	機能概要
D/A コンバータ	DAn_Set12BitsValue	ANO*n* 端子に出力するアナログ電圧値（12 ビット）を設定します。
タイマ	TAUm_Init	タイマ・アレイ・ユニットの機能を制御するうえで必要となる初期化処理を行います。
	TAUm_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。
	TAUm_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。
	TAUm_ChannelN_Start	チャネル *n* のカウントを開始します。
	TAUm_ChannelN_Stop	チャネル *n* のカウントを終了します。
	TAUm_ChannelN_ChangeCondition	カウント値を変更します。
	TAUm_ChannelN_ChangeTimerCondition	カウント値を変更します。
	TAUm_ChannelN_GetPulseWidth	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。
	TAUm_ChannelN_ChangeDuty	TO*mn* 端子に出力する PWM 信号のデューティ比を変更します。
	TAUm_ChannelN_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。
ウォッチドッグ・タイマ	WDT_Init	ウォッチドッグ・タイマの機能を制御するうえで必要となる初期化処理を行います。
	WDT_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。
	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。
リアルタイム・カウンタ	RTC_Init	リアルタイム・カウンタの機能を制御するうえで必要となる初期化処理を行います。
	RTC_UserInit	リアルタイム・カウンタに関するユーザ独自の初期化処理を行います。
	RTC_PowerOff	リアルタイム・カウンタに対するクロック供給を停止します。
	RTC_CounterEnable	リアルタイム・カウンタ（年，月，曜日，日，時，分，秒）のカウントを開始します。
	RTC_CounterDisable	リアルタイム・カウンタ（年，月，曜日，日，時，分，秒）のカウントを終了します。
	RTC_SetHourSystem	リアルタイム・カウンタの時間制（12 時間制，24 時間制）を設定します。
	RTC_CounterSet	リアルタイム・カウンタにカウント値（年，月，曜日，日，時，分，秒）を設定します。

周辺機能	API 関数名	機能概要
リアルタイム・カウンタ	RTC_CounterGet	リアルタイム・カウンタのカウント値（年，月，曜日，日，時，分，秒）を読み出します。
	RTC_ConstPeriodInterruptEnable	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。
	RTC_ConstPeriodInterruptDisable	定周期割り込み機能を終了します。
	RTC_ConstPeriodInterruptCallback	定周期割り込み INTRTC の発生に伴う処理を行います。
	RTC_AlarmEnable	アラーム割り込み機能を開始します。
	RTC_AlarmDisable	アラーム割り込み機能を終了します。
	RTC_AlarmSet	アラームの発生条件（曜日，時，分）を設定します。
	RTC_AlarmGet	アラームの発生条件（曜日，時，分）を読み出します。
	RTC_AlarmInterruptCallback	アラーム割り込み INTRTC の発生に伴う処理を行います。
	RTC_IntervalStart	インターバル割り込み機能を開始します。
	RTC_IntervalStop	インターバル割り込み機能を終了します。
	RTC_IntervalInterruptEnable	割り込み INTRTCI の発生周期を設定したのち，インターバル割り込み機能を開始します。
	RTC_IntervalInterruptDisable	インターバル割り込み機能を終了します。
	RTC_RTC1HZ_OutputEnable	RTC1HZ 端子に対するリアルタイム・カウンタ補正クロック（1 Hz）の出力を許可します。
	RTC_RTC1HZ_OutputDisable	RTC1HZ 端子に対するリアルタイム・カウンタ補正クロック（1 Hz）の出力を禁止します。
	RTC_RTCCL_OutputEnable	RTCCL 端子に対するリアルタイム・カウンタ・クロック（32 kHz 原発）の出力を許可します。
	RTC_RTCCL_OutputDisable	RTCCL 端子に対するリアルタイム・カウンタ・クロック（32 kHz 原発）の出力を禁止します。
	RTC_RTCDIV_OutputEnable	RTCDIV 端子に対するリアルタイム・カウンタ・クロック（32 kHz 分周）の出力を許可します。
	RTC_RTCDIV_OutputDisable	RTCDIV 端子に対するリアルタイム・カウンタ・クロック（32 kHz 分周）の出力を禁止します。
	RTC_ChangeCorrectionValue	時計誤差を補正するタイミング，および補正値を変更します。
クロック出力	PCL_Init	クロック出力制御回路の機能を制御するうえで必要となる初期化処理を行います。
	PCL_UserInit	クロック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。
	PCL_Start	クロック出力を開始します。
	PCL_Stop	クロック出力を停止します。
	PCL_ChangeFreq	PCL 端子への出力クロックを変更します。

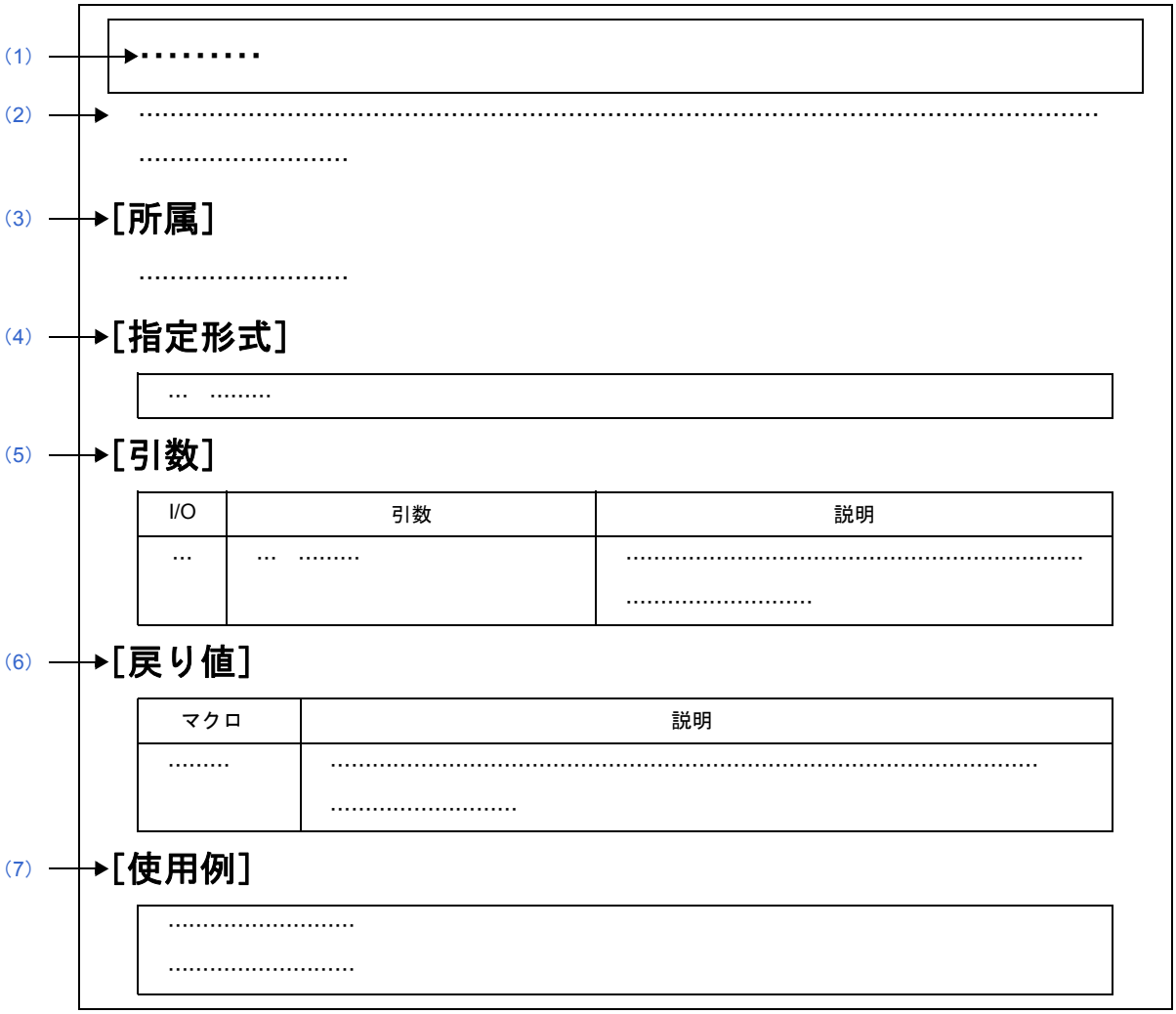
周辺機能	API 関数名	機能概要
クロック出力／ブザー出力	PCLBUZn_Init	クロック出力／ブザー出力制御回路の機能を制御するうえで必要となる初期化処理を行います。
	PCLBUZn_UserInit	クロック出力／ブザー出力制御回路に関するユーザ独自の初期化処理を行います。
	PCLBUZn_Start	クロック出力／ブザー出力を開始します。
	PCLBUZn_Stop	クロック出力／ブザー出力を停止します。
	PCLBUZn_ChangeFreq	PCLBUZ*n* 端子への出力クロックを変更します。
LCD コントローラ／ドライバ	LCD_Init	LCD コントローラ／ドライバの機能を制御するうえで必要となる初期化処理を行います。
	LCD_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。
	LCD_DisplayOn	LCD コントローラ／ドライバを表示オン状態にします。
	LCD_DisplayOff	LCD コントローラ／ドライバを表示オフ状態にします。
	LCD_VoltageOn	LCD コントローラ／ドライバの昇圧回路，および容量分割回路を動作許可としたのち，非選択信号をセグメント端子から出力します。
	LCD_VoltageOff	LCD コントローラ／ドライバの昇圧回路，および容量分割回路を動作停止としたのち，グランド・レベルの信号をセグメント端子／コモン端子に出力します。
DMA コントローラ	DMAn_Init	DMA コントローラの機能を制御するうえで必要となる初期化処理を行います。
	DMAn_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。
	DMAn_Enable	チャネル *n* を動作許可状態に設定します。
	DMAn_Disable	チャネル *n* を動作停止状態に設定します。
	DMAn_Hold	DMA 起動要求を保留します。
	DMAn_Restart	DMA 起動要求の保留を解除します。
	DMAn_CheckStatus	転送状態（転送終了，転送中）を読み出します。
	DMAn_SetData	転送先／転送元の RAM アドレス，およびデータの転送回数を設定します。
	DMAn_SoftwareTriggerOn	DMA 動作許可状態の際，DMA 転送を開始します。
低電圧検出回路	LVI_Init	低電圧検出回路の機能を制御するうえで必要となる初期化処理を行います。
	LVI_UserInit	低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。
	LVI_InterruptModeStart	低電圧検出動作を開始します（割り込み発生モード時）。

周辺機能	API 関数名	機能概要
低電圧検出回路	LVI_ResetModeStart	低電圧検出動作を開始します（内部リセット・モード時）。
	LVI_Stop	低電圧検出動作を停止します。
	LVI_SetLVILevel	低電圧検出レベルを設定します。

C. 3 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) [所属]

API 関数が出力される C ソース・ファイル名を示しています。

(4) [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。

(5) [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

(6) [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

(7) [使用例]

API 関数の使用例を示しています。

C. 3. 1 システム

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

表 C—2 システム用 API 関数

API 関数名	機能概要
CLOCK_Init	クロック発生回路の機能，オンチップ・デバッグ機能などを制御するうえで必要となる初期化処理を行います。
CLOCK_UserInit	クロック発生回路，オンチップ・デバッグなどに関するユーザ独自の初期化処理を行います。
CG_ReadResetSource	内部リセットの発生に伴う処理を行います。
CG_ChangeClockMode	CPU クロック／周辺ハードウエア・クロックを変更します。
CG_ChangeFrequency	CPU クロック／周辺ハードウエア・クロックの分周比を変更します。
CG_SelectPowerSaveMode	CPU のスタンバイ・モードを設定します。
CG_SelectStabTime	X1 クロックの発振安定時間を設定します。

CLOCK_Init

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

[所属]

CG_system.c

[指定形式]

```
void    CLOCK_Init ( void );
```

[引数]

なし

[戻り値]

なし

CLOCK_UserInit

クロック発生回路，オンチップ・デバッグなど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，CLOCK_Init のコールバック・ルーチンとして呼び出されます。

[所属]

CG_system_user.c

[指定形式]

```
void    CLOCK_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

CG_ReadResetSource

内部リセットの発生に伴う処理を行います。

[所属]

CG_system_user.c

[指定形式]

```
void    CG_ReadResetSource ( void );
```

[引数]

なし

[戻り値]

なし

[使用例]

以下に，リセット信号 RESET の発生要因別に異なる処理を実行する際の例を示します。

【CG_Systeminit.c】

```
void systeminit ( void ) {
    CG_ReadResetSource ();      /* リセット信号の発生要因別に処理を実行 */
    ……
}
```

【CG_system_user.c】

```
#include    "CG_macrodriver.h"
void CG_ReadResetSource ( void ) {
    UCHAR   flag = RESF;        /* リセット・コントロール・フラグ・レジスタ：RESF の内容確保 */
    if ( flag & 0x1 ) {         /* 発生要因の判別：LVIRF フラグのチェック */
        ……    /* 低電圧検出回路による内部リセット要求に対応した処理 */
    } else if ( flag & 0x10 ) { /* 発生要因の判別：WDRF フラグのチェック */
        ……    /* ウォッチドッグ・タイマによる内部リセット要求に対応した処理 */
    } else if ( flag & 0x80 ) { /* 発生要因の判別：TRAP フラグのチェック */
        ……    /* 不正命令の実行による内部リセット要求に対応した処理 */
    }
    ……
```

```
}
```

CG_ChangeClockMode

CPU クロック／周辺ハードウエア・クロックを変更します。

［所属］

CG_system.c

［指定形式］

```
#include    "CG_macrodriver.h"
#include    "CG_system.h"
MD_STATUS   CG_ChangeClockMode ( enum ClockMode mode );
```

［引数］

I/O	引数	説明
I	enum ClockMode *mode*;	CPU クロック／周辺ハードウエア・クロックの種類 【Fx3】 HIOCLK : 高速内蔵発振クロック SYSX1CLK : X1 クロック SYSEXTCLK : 外部メイン・システム・クロック FILCLK : 低速内蔵発振クロック 【Ix3】 HIOCLK : 高速内蔵発振クロック HIO40CLK : 40MHz 高速内蔵発振クロック SYSX1CLK : X1 クロック SYSEXTCLK : 外部メイン・システム・クロック SUBCLK : サブシステム・クロック 【Kx3】 HIOCLK : 高速内蔵発振クロック SYSX1CLK : X1 クロック SYSEXTCLK : 外部メイン・システム・クロック SUBCLK : サブシステム・クロック 【Kx3-A】【Kx3-L】【Lx3】 HIOCLK : 高速内蔵発振クロック HIO20CLK : 20MHz 高速内蔵発振クロック SYSX1CLK : X1 クロック SYSEXTCLK : 外部メイン・システム・クロック SUBCLK : 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【Fx3】【Kx3】 - X1 クロックへの変更はできません。 異常終了【Ix3】 - 40MHz 高速内蔵発振クロックへの変更はできません。 異常終了【Kx3-A】【Kx3-L】【Lx3】 - 20MHz 高速内蔵発振クロックへの変更はできません。
MD_ERROR2	異常終了【Fx3】【Kx3】 - 外部メイン・システム・クロックへの変更はできません。 異常終了【Ix3】【Kx3-A】【Kx3-L】【Lx3】 - X1 クロックへの変更はできません。
MD_ERROR3	異常終了【Fx3】 - 低速内蔵発振クロックへの変更はできません。 異常終了【Ix3】【Kx3-A】【Kx3-L】【Lx3】 - 外部メイン・システム・クロックへの変更はできません。 異常終了【Kx3】 - XT1，XT2 端子が入力モードのため，サブシステム・クロックへの変更はできません。
MD_ERROR4	異常終了【Ix3】【Kx3-A】【Kx3-L】【Lx3】 - サブシステム・クロックへの変更はできません。
MD_ARGERROR	引数の指定が不正

CG_ChangeFrequency

CPU クロック／周辺ハードウエア・クロックの分周比を変更します。

［所属］

CG_system.c

［指定形式］

```
#include    "CG_macrodriver.h"
#include    "CG_system.h"
MD_STATUS   CG_ChangeFrequency ( enum CPUClock clock );
```

［引数］

I/O	引数	説明
I	enum CPUClock *clock*;	分周比の種類 【Fx3】 SYSTEMCLOCK : fPLL SYSONEHALF : fPLL/2 SYSONEFOURTH : fPLL/4 SYSONEEIGHTH : fPLL/8 SYSONESIXTEENTH : fPLL/16 SYSONETHIRTYSECOND : fPLL/32 【Ix3】【Kx3】【Kx3-L】 SYSTEMCLOCK : fMAIN SYSONEHALF : fMAIN/2 SYSONEFOURTH : fMAIN/4 SYSONEEIGHTH : fMAIN/8 SYSONESIXTEENTH : fMAIN/16 SYSONETHIRTYSECOND : fMAIN/32 【Kx3-A】【Lx3】 SYSTEMCLOCK : fMAIN SYSONEHALF : fMAIN/2 SYSONEFOURTH : fMAIN/4 SYSONEEIGHTH : fMAIN/8 SYSONESIXTEENTH : fMAIN/16 SYSONETHIRTYSECOND : fMAIN/32 SUB : fSUB SUBONEHALF : fSUB/2

備考 fPLL は PLL クロックの周波数を，fMAIN はメイン・システム・クロックの周波数を，fSUB はサブシステム・クロックの周波数を意味します。

［戻り値］

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

CG_SelectPowerSaveMode

CPU のスタンバイ・モードを設定します。

[所属]

CG_system.c

[指定形式]

```
#include    "CG_macrodriver.h"
#include    "CG_system.h"
MD_STATUS   CG_SelectPowerSaveMode ( enum PSLevel level );
```

[引数]

I/O	引数	説明
I	enum PSLevel *level*;	スタンバイ・モードの種類 PSSTOP : STOP モード PSHALT : HALT モード

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR	異常終了 - CPU がサブシステム・クロック（XT1 発振回路）で動作している場合，STOP モードを指定することはできません。
MD_ARGERROR	引数の指定が不正

[使用例]

以下に，スタンバイ・モードを“STOP モード”へと移行させる際の例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
#include    "CG_system.h"
void main ( void ) {
    MD_STATUS   ret;
    ……
    TAU0_PowerOff ();                          /* クロック供給の停止 */
```

```
    ret = CG_SelectPowerSaveMode ( PSSTOP );    /* STOP モードへの移行 */
    if ( ret != MD_OK ) {
        while ( 1 );
    }
    TAU0_Init ();                               /* タイマ・アレイ・ユニットの初期化 */
    TAU0_Channel0_Start ();                     /* チャネル 0 のカウント開始 */
    ……
}
```

CG_SelectStabTime

X1 クロックの発振安定時間を設定します。

[所属]

CG_system.c

[指定形式]

```
#include    "CG_macrodriver.h"
#include    "CG_system.h"
MD_STATUS   CG_SelectStabTime ( enum StabTime waittime );
```

[引数]

I/O	引数	説明
I	enum StabTime *waittime*;	発振安定時間の種類 STLEVEL0： 2^8/fx STLEVEL1： 2^9/fx STLEVEL2： 2^10/fx STLEVEL3： 2^11/fx STLEVEL4： 2^13/fx STLEVEL5： 2^15/fx STLEVEL6： 2^17/fx STLEVEL7： 2^18/fx

備考 fx は，X1 クロックの周波数を意味します。

[戻り値]

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

C. 3. 2 外部バス

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

表 C—3 外部バス用 API 関数

API 関数名	機能概要
BUS_Init	外部バス・インタフェースの機能（外部バスを内蔵 ROM，RAM，SFR 以外の領域に接続する機能）を制御するうえで必要となる初期化処理を行います。
BUS_UserInit	外部バス・インタフェースに関するユーザ独自の初期化処理を行います。
BUS_PowerOff	外部バス・インタフェースに対するクロック供給を停止します。

BUS_Init

外部バス・インタフェースの機能（外部バスを内蔵 ROM，RAM，SFR 以外の領域に接続する機能）を制御するうえで必要となる初期化処理を行います。

[所属]

CG_bus.c

[指定形式]

```
void    BUS_Init ( void );
```

[引数]

なし

[戻り値]

なし

BUS_UserInit

外部バス・インタフェースに関するユーザ独自の初期化処理を行います。

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

[所属]

CG_bus_user.c

[指定形式]

```
void    BUS_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

BUS_PowerOff

外部バス・インタフェースに対するクロック供給を停止します。

備考 本 API 関数の呼び出しにより，外部バス・インタフェース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（メモリ拡張モード制御レジスタ：MEM など）への書き込みは無視されます。

[所属]

CG_bus.c

[指定形式]

```
void    BUS_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

C. 3. 3 ポート

以下に，コード生成がポート用として出力する API 関数の一覧を示します。

表 C—4 ポート用 API 関数

API 関数名	機能概要
PORT_Init	ポートの機能を制御するうえで必要となる初期化処理を行います。
PORT_UserInit	ポートに関するユーザ独自の初期化処理を行います。
PORT_ChangePmnInput	端子の入出力モードを出力モードから入力モードへと切り替えます。
PORT_ChangePmnOutput	端子の入出力モードを入力モードから出力モードへと切り替えます。

PORT_Init

ポートの機能を制御するうえで必要となる初期化処理を行います。

[所属]

CG_port.c

[指定形式]

```
void    PORT_Init ( void );
```

[引数]

なし

[戻り値]

なし

PORT_UserInit

ポートに関するユーザ独自の初期化処理を行います。

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

[所属]

CG_port_user.c

[指定形式]

```
void    PORT_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

PORT_ChangeP*mn*Input

端子の入出力モードを出力モードから入力モードへと切り替えます。

[所属]

CG_port.c

[指定形式]

本 API 関数の指定形式は，対象端子に内蔵プルアップ抵抗／ TTL 入力バッファが存在するか否かにより異なります。

- 内蔵プルアップ抵抗：なし，TTL 入力バッファ：なし

```
void    PORT_ChangeP mn Input ( void );
```

- 内蔵プルアップ抵抗：あり，TTL 入力バッファ：なし

```
#include    "CG_macrodriver.h"
void    PORT_ChangeP mn Input ( BOOL enablepu );
```

- 内蔵プルアップ抵抗：あり，TTL 入力バッファ：あり

```
#include    "CG_macrodriver.h"
void    PORT_ChangeP mn Input ( BOOL enablepu, BOOL enablettl );
```

備考 *mn* は，ポート番号を意味します。

[引数]

I/O	引数	説明
I	BOOL *enablepu*;	内蔵プルアップ抵抗の使用有無 MD_TRUE： 使用する MD_FALSE： 使用しない
I	BOOL *enablettl*;	入力バッファの種類 MD_TRUE： TTL 入力バッファ MD_FALSE： 通常入力バッファ

[戻り値]

なし

[使用例 1]

以下に，P00 端子（内蔵プルアップ抵抗：あり，TTL 入力バッファ：なし）を
　入出力モードの種類：　　入力モード
　内蔵プルアップ抵抗の使用有無：　使用する
に変更する際の例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
void main ( void ) {
    .........
    PORT_ChangeP00Input ( MD_TRUE );    /* 入出力モードの切り替え */
    .........
}
```

[使用例 2]

以下に，P00 端子（内蔵プルアップ抵抗：あり，TTL 入力バッファ：なし）を
　入出力モードの種類：　　入力モード
　内蔵プルアップ抵抗の使用有無：　使用しない
に変更する際の例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
void main ( void ) {
    .........
    PORT_ChangeP00Input ( MD_FALSE );   /* 入出力モードの切り替え */
    .........
}
```

[使用例 3]

以下に，P04 端子（内蔵プルアップ抵抗：あり，TTL 入力バッファ：あり）を
　入出力モードの種類：　　入力モード
　内蔵プルアップ抵抗の使用有無：　使用しない
　入力バッファの種類：　　TTL 入力バッファ
に変更する際の例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
void main ( void ) {
```

```
    .........
    PORT_ChangeP04Input ( MD_FALSE, MD_TRUE );  /* 入出力モードの切り替え */
    .........
}
```

PORT_ChangeP*mn*Output

端子の入出力モードを入力モードから出力モードへと切り替えます。

［所属］

CG_port.c

［指定形式］

対象デバイスが 78K0R/Fx3 の場合，本 API 関数の指定形式は，対象端子で N-ch オープン・ドレーン出力が行われるか否か，スロー・モードの指定を行うか否かにより異なります。

- N-ch オープン・ドレーン出力：なし，スロー・モード：なし【Fx3】

```
#include    "CG_macrodriver.h"
void    PORT_ChangeP mn Output ( BOOL initialvalue );
```

- N-ch オープン・ドレーン出力：あり，スロー・モード：なし【Fx3】

```
#include    "CG_macrodriver.h"
void    PORT_ChangeP mn Output ( BOOL enablench, BOOL initialvalue );
```

- N-ch オープン・ドレーン出力：なし，スロー・モード：あり【Fx3】

```
#include    "CG_macrodriver.h"
void    PORT_ChangeP mn Output ( BOOL enableslow, BOOL initialvalue );
```

- N-ch オープン・ドレーン出力：あり，スロー・モード：あり【Fx3】

```
#include    "CG_macrodriver.h"
void    PORT_ChangeP mn Output ( BOOL enablench, BOOL enableslow, BOOL initialvalue );
```

また，対象デバイスが 78K0R/Ix3，78K0R/Kx3，78K0R/Kx3-A，78K0R/Kx3-L，または 78K0R/Lx3 の場合，本 API 関数の指定形式は，対象端子で N-ch オープン・ドレーン出力が行われるか否かにより異なります。

- N-ch オープン・ドレーン出力：なし【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

```
#include    "CG_macrodriver.h"
void    PORT_ChangeP mn Output ( BOOL initialvalue );
```

- N-ch オープン・ドレーン出力：あり【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

```
#include    "CG_macrodriver.h"
void    PORT_ChangePmnOutput ( BOOL enablench, BOOL initialvalue );
```

備考 *nm* は，ポート番号を意味します。

[引数]

I/O	引数	説明
I	BOOL *enablench*;	出力モードの種類 MD_TRUE： N-ch オープン・ドレーン出力（VDD 耐圧）モード MD_FALSE： 通常出力モード
I	BOOL *enableslow*;	出力モードの種類 MD_TRUE： スロー・モード MD_FALSE： 通常モード
I	BOOL *initialvalue*;	初期出力値 MD_SET： High レベル“1”を出力 MD_CLEAR： Low レベル“0”を出力

[戻り値]

なし

[使用例 1]

以下に，P00 端子（N-ch オープン・ドレーン出力：なし）を

入出力モードの種類： 出力モード

初期出力値： High レベル“1”を出力

に変更する際の例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
void main ( void ) {
    .........
    PORT_ChangeP00Output ( MD_SET );    /* 入出力モードの切り替え */
    .........
}
```

[使用例 2]

以下に，P04 端子（N-ch オープン・ドレーン出力：あり）を

入出力モードの種類： 出力モード

出力モードの種類： N-ch オープン・ドレーン出力（VDD 耐圧）モード

初期出力値： Low レベル“0”を出力

に変更する際の例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
void main ( void ) {
    .........
    PORT_ChangeP04Output ( MD_TRUE, MD_CLEAR ); /* 入出力モードの切り替え */
    .........
}
```

C. 3. 4 割り込み

以下に，コード生成が割り込み用として出力するAPI関数の一覧を示します。

表C—5 割り込み用API関数

API関数名	機能概要
INTP_Init	外部割り込みINTP*n*の機能を制御するうえで必要となる初期化処理を行います。
INTP_UserInit	外部割り込みINTP*n*に関するユーザ独自の初期化処理を行います。
KEY_Init	キー割り込みINTKRの機能を制御するうえで必要となる初期化処理を行います。
KEY_UserInit	キー割り込みINTKRに関するユーザ独自の初期化処理を行います。
INT_MaskableInterruptEnable	マスカブル割り込みの受け付けを禁止／許可します。
INTPn_Disable	マスカブル割り込み（外部割込み要求）INTP*n*の受け付けを禁止します。
INTPn_Enable	マスカブル割り込み（外部割込み要求）INTP*n*の受け付けを許可します。
KEY_Disable	キー割り込みINTKRの受け付けを禁止します。
KEY_Enable	キー割り込みINTKRの受け付けを許可します。

INTP_Init

外部割り込み INTP*n* の機能を制御するうえで必要となる初期化処理を行います。

[所属]

CG_int.c

[指定形式]

```
void    INTP_Init ( void );
```

[引数]

なし

[戻り値]

なし

INTP_UserInit

外部割り込み INTP*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，INTP_Init のコールバック・ルーチンとして呼び出されます。

[所属]

CG_int_user.c

[指定形式]

```
void    INTP_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

KEY_Init

キー割り込み INTKR の機能を制御するうえで必要となる初期化処理を行います。

[所属]

CG_int.c

[指定形式]

```
void    KEY_Init ( void );
```

[引数]

なし

[戻り値]

なし

KEY_UserInit

キー割り込み INTKR 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，KEY_Init のコールバック・ルーチンとして呼び出されます。

[所属]

CG_int_user.c

[指定形式]

```
void    KEY_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

INT_MaskableInterruptEnable

マスカブル割り込みの受け付けを禁止／許可します。

[所属]

CG_int.c

[指定形式]

- 【Fx3】【Ix3】【Kx3-A】【Kx3-L】【Lx3】

```
#include    "CG_macrodriver.h"
#include    "CG_int.h"
MD_STATUS   INT_MaskableInterruptEnable ( enum MaskableSource name, BOOL enableflag );
```

- 【Kx3】

```
#include    "CG_macrodriver.h"
#include    "CG_int.h"
void    INT_MaskableInterruptEnable ( enum MaskableSource name, BOOL enableflag );
```

[引数]

I/O	引数	説明
I	enum MaskableSource *name*;	マスカブル割り込みの種類 INT_*xxx*： マスカブル割り込み
I	BOOL *enableflag*;	受け付けの禁止／許可 MD_TRUE： 受け付けを許可 MD_FALSE： 受け付けを禁止

備考 マスカブル割り込みの種類 INT_*xxx* についての詳細は，ヘッダ・ファイル CG_int.h を参照してください。

[戻り値]

- 【Fx3】【Ix3】【Kx3-A】【Kx3-L】【Lx3】

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

- 【Kx3】

なし

[使用例 1]

以下に，マスカブル割り込み INTP0 の受け付けを“禁止”に設定する際の例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
#include    "CG_int.h"
void main ( void ) {
    .........
    INT_MaskableInterruptEnable ( INT_INTP0, MD_FALSE );  /* マスカブル割り込み INTP0 の受け付け禁止
*/
    .........
}
```

[使用例 2]

以下に，マスカブル割り込み INTP0 の受け付けを“許可”に設定する際の例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
#include    "CG_int.h"
void main ( void ) {
    .........
    INT_MaskableInterruptEnable ( INT_INTP0, MD_TRUE );   /* マスカブル割り込み INTP0 の受け付け許可
*/
    .........
}
```

INTP*n*_Disable

マスカブル割り込み（外部割り込み要求）INTP*n* の受け付けを禁止します。

[所属]

CG_int.c

[指定形式]

```
void    INTPn_Disable ( 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

INTP*n*_Enable

マスカブル割り込み（外部割り込み要求）INTP*n* の受け付けを許可します。

[所属]

CG_int.c

[指定形式]

```
void    INTPn_Enable ( 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

KEY_Disable

キー割り込み INTKR の受け付けを禁止します。

[所属]

CG_int.c

[指定形式]

```
void    KEY_Disable ( void );
```

[引数]

なし

[戻り値]

なし

KEY_Enable

キー割り込み INTKR の受け付けを許可します。

[所属]

CG_int.c

[指定形式]

```
void    KEY_Enable ( void );
```

[引数]

なし

[戻り値]

なし

C. 3. 5 シリアル

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

表 C—6 シリアル用 API 関数

API 関数名	機能概要
SAUm_Init	シリアル・アレイ・ユニット，およびシリアル・インタフェースの機能を制御するうえで必要となる初期化処理を行います。
SAUm_UserInit	シリアル・アレイ・ユニット，およびシリアル・インタフェースに関するユーザ独自の初期化処理を行います。
SAUm_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。
UARTn_Init	シリアル・インタフェース（UART）用チャネルの初期化処理を行います。
UARTn_Start	UART 通信を待機状態にします。
UARTn_Stop	UART 通信を終了します。
UARTn_SendData	データの UART 送信を開始します。
UARTn_ReceiveData	データの UART 受信を開始します。
UARTn_SendEndCallback	UART 送信完了割り込み INTST*n* の発生に伴う処理を行います。
UARTn_ReceiveEndCallback	UART 受信完了割り込み INTSR*n* の発生に伴う処理を行います。
UARTn_SoftOverRunCallback	UART 受信完了割り込み INTSR*n* の発生に伴う処理を行います。
UARTn_ErrorCallback	UART 通信におけるエラー割り込み INTSRE*n* の発生に伴う処理を行います。
CSImn_Init	シリアル・インタフェース（CSI）用チャネルの初期化処理を行います。
CSImn_Start	CSI 通信を待機状態にします。
CSImn_Stop	CSI 通信を終了します。
CSImn_SendData	データの CSI 送信を開始します。
CSImn_ReceiveData	データの CSI 受信を開始します。
CSImn_SendReceiveData	データの CSI 送受信を開始します。
CSImn_SendEndCallback	CSI 通信完了割り込み INTCSI*mn* の発生に伴う処理を行います。
CSImn_ReceiveEndCallback	CSI 通信完了割り込み INTCSI*mn* の発生に伴う処理を行います。
CSImn_ErrorCallback	CSI 通信におけるエラー割り込み INTSRE*n* の発生に伴う処理を行います。
IICmn_Init	シリアル・インタフェース（簡易 IIC）用チャネルの初期化処理を行います。
IICmn_Stop	簡易 IIC 通信を終了します。
IICmn_MasterSendStart	簡易 IIC マスタ送信を開始します。
IICmn_MasterReceiveStart	簡易 IIC マスタ受信を開始します。
IICmn_StartCondition	スタート・コンディションを発生します。
IICmn_StopCondition	ストップ・コンディションを発生します。
IICmn_MasterSendEndCallback	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。
IICmn_MasterReceiveEndCallback	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。
IICmn_MasterErrorCallback	簡易 IIC 通信におけるパリティ・エラー（ACK エラー）の検出に伴う処理を行います。

API 関数名	機能概要
UARTFn_Init	シリアル・インタフェース（UARTF*n*）の初期化処理を行います。
UARTFn_PowerOff	シリアル・インタフェース（UARTF*n*）に対するクロック供給を停止します。
UARTFn_Start	UARTF 通信を待機状態にします。
UARTFn_Stop	UARTF 通信を終了します。
UARTFn_SendData	データの UARTF 送信を開始します。
UARTFn_ReceiveData	データの UARTF 受信を終了します。
UARTFn_SetComparisonData	受信データと比較するデータを設定します。
UARTFn_DataComparisonEnable	データの比較を開始します。
UARTFn_DataComparisonDisable	データの比較を終了します。
UARTFn_SendEndCallback	送信割り込み INTLT*n* の発生に伴う処理を行います。
UARTFn_ReceiveEndCallback	受信完了割り込み INTLR*n* の発生に伴う処理を行います。
UARTFn_SoftOverRunCallback	受信完了割り込み INTLR*n* の発生に伴う処理を行います。
UARTFn_ExpBitCetectCallback	ステータス割り込み INTLS*n* の発生に伴う処理を行います。
UARTFn_IDMatchCallback	ステータス割り込み INTLS*n* の発生に伴う処理を行います。
UARTFn_ErrorCallback	ステータス割り込み INTLS*n* の発生に伴う処理を行います。
IICA_Init	シリアル・インタフェース（IICA）の初期化処理を行います。
IICA_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。
IICA_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。
IICA_Stop	IICA 通信を終了します。
IICA_MasterSendStart	IICA マスタ送信を開始します。
IICA_MasterReceiveStart	IICA マスタ受信を開始します。
IICA_StopCondition	ストップ・コンディションを発生します。
IICA_MasterSendEndCallback	IICA 通信完了割り込み INTIICA の発生に伴う処理を行います。
IICA_MasterReceiveEndCallback	IICA 通信完了割り込み INTIICA の発生に伴う処理を行います。
IICA_MasterErrorCallback	IICA マスタ通信におけるエラーの検出に伴う処理を行います。
IICA_SlaveSendStart	IICA スレーブ送信を開始します。
IICA_SlaveReceiveStart	IICA スレーブ受信を開始します。
IICA_SlaveSendEndCallback	IICA 通信完了割り込み INTIICA の発生に伴う処理を行います。
IICA_SlaveReceiveEndCallback	IICA 通信完了割り込み INTIICA の発生に伴う処理を行います。
IICA_SlaveErrorCallback	IICA スレーブ通信におけるエラーの検出に伴う処理を行います。
IICA_GetStopConditionCallback	IICA スレーブ通信におけるストップ・コンディションの検出に伴う処理を行います。
IICn_Init	シリアル・インタフェース（IIC*n*）の初期化処理を行います。
IICn_UserInit	シリアル・インタフェース（IIC*n*）に関するユーザ独自の初期化処理を行います。
IICn_Stop	IIC*n* 通信を終了します。

API 関数名	機能概要
IICn_MasterSendStart	IIC*n* マスタ送信を開始します。
IICn_MasterReceiveStart	IIC*n* マスタ受信を開始します。
IICn_MasterSendEndCallback	IIC*n* 通信完了割り込み INTIIC*n* の発生に伴う処理を行います。
IICn_MasterReceiveEndCallback	IIC*n* 通信完了割り込み INTIIC*n* の発生に伴う処理を行います。
IICn_MasterErrorCallback	IIC*n* マスタ通信におけるエラーの検出に伴う処理を行います。
IICn_SlaveSendStart	IIC*n* スレーブ送信を開始します。
IICn_SlaveReceiveStart	IIC*n* スレーブ受信を開始します。
IICn_SlaveSendEndCallback	IIC*n* 通信完了割り込み INTIIC*n* の発生に伴う処理を行います。
IICn_SlaveReceiveEndCallback	IIC*n* 通信完了割り込み INTIIC*n* の発生に伴う処理を行います。
IICn_SlaveErrorCallback	IIC*n* スレーブ通信におけるエラーの検出に伴う処理を行います。
IICn_GetStopConditionCallback	IIC*n* スレーブ通信におけるストップ・コンディションの検出に伴う処理を行います。

SAU*m*_Init

シリアル・アレイ・ユニット，およびシリアル・インタフェースの機能を制御するうえで必要となる初期化処理を行います。

[所属]

CG_serial.c

[指定形式]

```
void    SAUm_Init ( 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

SAU*m*_UserInit

シリアル・アレイ・ユニット，およびシリアル・インタフェースに関するユーザ独自の初期化処理を行います。

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

[所属]

CG_serial_user.c

[指定形式]

```
void    SAUm_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

SAU*m*_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。

[所属]

CG_serial.c

[指定形式]

```
void    SAUm_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

UART*n*_Init

シリアル・インタフェース（UART）用チャネルの初期化処理を行います。

備考 本 API 関数は，SAUm_Init 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。

[所属]

CG_serial.c

[指定形式]

```
void    UARTn_Init ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

UART*n*_Start

UART 通信を待機状態にします。

[所属]

CG_serial.c

[指定形式]

```
void    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

UART*n*_Stop

UART 通信を終了します。

[所属]

CG_serial.c

[指定形式]

```
void    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

UART*n*_SendData

データの UART 送信を開始します。

備考 1. 本 API 関数では，引数 *txbuf* 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 *txnum* 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 UARTn_Start を呼び出す必要があります。

[所属]

CG_serial.c

[指定形式]

```
#include    “CG_macrodriver.h“
MD_STATUS   UARTn_SendData ( UCHAR *txbuf, USHORT txnum );
```

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	UCHAR *txbuf;	送信するデータを格納したバッファへのポインタ
I	USHORT txnum;	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

[使用例]

以下に，チャネル 0 から 4 バイトの固定長データを 1 回だけ UART 送信する際の例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
BOOL    gFlag;                          /* 送信完了フラグ */
void main ( void ) {
    UCHAR   txbuf[] = "ABCD";
```

```
    USHORT  txnum = 4;
    gFlag = 1;                          /* 送信完了フラグの初期化 */
    ……
    UART0_Start ();                     /* UART 通信の開始 */
    UART0_SendData ( &txbuf, txnum );   /* UART 送信の開始 */
    while ( gFlag );                    /* txnum 個の送信待ち */
    ……
}
```

【CG_serial_user.c】

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                  /* 送信完了フラグ */
__interrupt void MD_INTST0 ( void ) {   /* 割り込み INTST0 発生時の割り込み処理 */
    if ( gUart0TxCnt > 0 ) {
        ……
    } else {
        UART0_SendEndCallback ();       /* コールバック・ルーチンの呼び出し */
    }
}

void UART0_SendEndCallback ( void ) {   /* 割り込み INTST0 発生時のコールバック・ルーチン */
    gFlag = 0;                          /* 送信完了フラグの設定 */
}
```

UART*n*_ReceiveData

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 *rxnum* で指定された回数だけ繰り返し行い，引数 *rxbuf* 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，UARTn_Start を呼び出すことにより開始されます。

[所属]

CG_serial.c

[指定形式]

```
#include    "CG_macrodriver.h"
MD_STATUS   UARTn_ReceiveData ( UCHAR *rxbuf, USHORT rxnum );
```

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	UCHAR **rxbuf*;	受信したデータを格納するバッファへのポインタ
I	USHORT *rxnum*;	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

[使用例]

以下に，チャネル 0 から 4 バイトの固定長データを 1 回だけ UART 受信する際の例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
BOOL    gFlag;                              /* 受信完了フラグ */
void main ( void ) {
    UCHAR   rxbuf[10];
```

```
    USHORT  rxnum = 4;
    gFlag = 1;                              /* 受信完了フラグの初期化 */
    .........
    UART0_ReceiveData ( &rxbuf, rxnum );    /* UART 受信の開始 */
    UART0_Start ();                         /* UART 通信の開始 */
    while ( gFlag );                        /* rxnum 個の受信待ち */
    .........
}
```

【CG_serial_user.c】

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                      /* 受信完了フラグ */
__interrupt void MD_INTSR0 ( void ) {       /* 割り込み INTSR0 発生時の割り込み処理 */
    .........
    if ( gUart0RxLen > gUart0RxCnt ) {
        .........
        if ( gUart0RxLen == gUart0RxCnt ) {
            UART0_ReceiveEndCallback ();    /* コールバック・ルーチンの呼び出し */
        }
    }
}

void UART0_ReceiveEndCallback ( void ) {    /* 割り込み INTSR0 発生時のコールバック・ルーチン */
    gFlag = 0;                              /* 受信完了フラグの設定 */
}
```

UART*n*_SendEndCallback

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 MD_INTST*n* のコールバック・ルーチン（UARTn_SendData の引数 *txnum* 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。

[所属]

CG_serial_user.c

[指定形式]

```
void    UARTn_SendEndCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

UART*n*_ReceiveEndCallback

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 MD_INTSR*n* のコールバック・ルーチン（UARTn_ReceiveData の引数 *rxnum* 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。

[所属]

CG_serial_user.c

[指定形式]

```
void    UARTn_ReceiveEndCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

UART*n*_SoftOverRunCallback

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 MD_INTSR*n* のコールバック・ルーチン（UARTn_ReceiveData の引数 *rxnum* 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。

[所属]

CG_serial_user.c

[指定形式]

- 【Fx3】

```
void    UARTn_SoftOverRunCallback ( void );
```

- 【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

```
#include    "CG_ad.h"
void    UARTn_SoftOverRunCallback ( UCHAR rx_data );
```

備考 *n* は，チャネル番号を意味します。

[引数]

- 【Fx3】

なし

- 【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

I/O	引数	説明
O	UCHAR *rx_data*;	受信したデータ（UARTn_ReceiveData の引数 *rxnum* で指定された数以上に受信したデータ）

[戻り値]

なし

UART*n*_ErrorCallback

UART 通信におけるエラー割り込み INTSRE*n* の発生に伴う処理を行います。

備考 本 API 関数は，エラー割り込み INTSRE*n* に対応した割り込み処理 MD_INTSRE*n* のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    UARTn_ErrorCallback ( UCHAR 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	UCHAR *err_type*;	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

[使用例]

以下に，エラー割り込みの発生要因別にコールバック処理を行う際の例を示します。

【CG_serial_user.c】

```
#include    "CG_macrodriver.h"
__interrupt void MD_INTSRE0 ( void ) {        /* 割り込み INTSRE0 発生時の割り込み処理 */
    UCHAR   err_type;
    ……
    UART0_ErrorCallback ( err_type );         /* コールバック・ルーチンの呼び出し */
}
```

```
void UART0_ErrorCallback ( UCHAR err_type ) { /* 割り込み INTSRE0 発生時のコールバック・ルーチン */
    if ( err_type & 0x1 ) {                   /* 発生要因の判別 */
        ·····   /* オーバラン・エラーが発生した際のコールバック処理 */
    } else if ( err_type & 0x2 ) {            /* 発生要因の判別 */
        ·····   /* パリティ・エラーが発生した際のコールバック処理 */
    } else if ( err_type & 0x4 ) {            /* 発生要因の判別 */
        ·····   /* フレーミング・エラーが発生した際のコールバック処理 */
    }
}
```

CSI*mn*_Init

シリアル・インタフェース（CSI）用チャネルの初期化処理を行います。

備考 本 API 関数は，SAUm_Init 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。

[所属]

CG_serial.c

[指定形式]

```
void    CSImn_Init ( 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

CSI*mn*_Start

CSI 通信を待機状態にします。

[所属]

CG_serial.c

[指定形式]

```
void    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

CSI*mn*_Stop

CSI 通信を終了します。

[所属]

CG_serial.c

[指定形式]

```
void    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

CSI*mn*_SendData

データの CSI 送信を開始します。

備考 1. 本 API 関数では，引数 *txbuf* 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 *txnum* 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 CSImn_Start を呼び出す必要があります。

[所属]

CG_serial.c

[指定形式]

```
#include    "CG_macrodriver.h"
MD_STATUS   CSImn_SendData ( UCHAR *txbuf, USHORT txnum );
```

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	UCHAR *txbuf;	送信するデータを格納したバッファへのポインタ
I	USHORT txnum;	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

[使用例]

以下に，チャネル 00 から 4 バイトの固定長データを 1 回だけ CSI 送信する際の例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
BOOL    gFlag;                          /* 送信完了フラグ */
void main ( void ) {
    UCHAR   txbuf[] = "ABCD";
```

```
    USHORT  txnum = 4;
    gFlag = 1;                          /* 送信完了フラグの初期化 */
    ……
    CSI00_Start ();                     /* CSI 通信の開始 */
    CSI00_SendData ( &txbuf, txnum );   /* CSI 送信の開始 */
    while ( gFlag );                    /* txnum 個の送信待ち */
    ……
}
```

【CG_serial_user.c】

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                  /* 送信完了フラグ */
__interrupt void MD_INTCSI00 ( void ) {   /* 割り込み INTCSI00 発生時の割り込み処理 */
    if ( gCsi00TxCnt > 0 ) {
        ……
    } else {
        CSI00_SendEndCallback ();       /* コールバック・ルーチンの呼び出し */
    }
}

void CSI00_SendEndCallback ( void ) {     /* 割り込み INTCSI00 発生時のコールバック・ルーチン */
    gFlag = 0;                          /* 送信完了フラグの設定 */
}
```

CSI*mn*_ReceiveData

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 *rxnum* で指定された回数だけ繰り返し行い，引数 *rxbuf* 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 CSImn_Start を呼び出す必要があります。

[所属]

CG_serial.c

[指定形式]

```
#include    "CG_macrodriver.h"
MD_STATUS   CSImn_ReceiveData ( UCHAR *rxbuf, USHORT rxnum );
```

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	UCHAR **rxbuf*;	受信したデータを格納するバッファへのポインタ
I	USHORT *rxnum*;	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

[使用例]

以下に，チャネル 00 から 4 バイトの固定長データを 1 回だけ CSI 受信する際の例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
BOOL    gFlag;                              /* 受信完了フラグ */
void main ( void ) {
    UCHAR   rxbuf[10];
```

```
    USHORT  rxnum = 4;
    gFlag = 1;                              /* 受信完了フラグの初期化 */
    ……
    CSI00_Start ();                         /* CSI 通信の開始 */
    CSI00_ReceiveData ( &rxbuf, rxnum );    /* CSI 受信の開始 */
    while ( gFlag );                        /* rxnum 個の受信待ち */
    ……
}
```

【CG_serial_user.c】

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                      /* 受信完了フラグ */
__interrupt void MD_INTCSI00 ( void ) {     /* 割り込み INTCSI00 発生時の割り込み処理 */
    if ( gCsi00RxCnt < gCsi00RxLen ) {
        ……
        if ( gCsi00RxCnt == gCsi00RxLen ) {
            CSI00_ReceiveEndCallback ();    /* コールバック・ルーチンの呼び出し */

        } else {
            ……
        }
    }
}

void CSI00_ReceiveEndCallback ( void ) {    /* 割り込み INTCSI00 発生時のコールバック・ルーチン */
    gFlag = 0;                              /* 受信完了フラグの設定 */
}
```

CSI*mn*_SendReceiveData

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 *txbuf* 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 *txnum* 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 *txnum* で指定された回数だけ繰り返し行い，引数 *rxbuf* 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 CSImn_Start を呼び出す必要があります。

[所属]

CG_serial.c

[指定形式]

```
#include    "CG_macrodriver.h"
MD_STATUS   CSImn_SendReceiveData ( UCHAR *txbuf, USHORT txnum, UCHAR *rxbuf );
```

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	UCHAR *txbuf;	送信するデータを格納したバッファへのポインタ
I	USHORT txnum;	送受信するデータの総数
O	UCHAR *rxbuf;	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

[使用例]

以下に，チャネル 00 から 4 バイトの固定長データを 1 回だけ CSI 送受信する際の例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
BOOL    gSflag;                                         /* 送信完了フラグ */
void main ( void ) {
    UCHAR   txbuf[] = "0123";
    USHORT  txnum = 4;
    UCHAR   rxbuf[10];
    gSflag = 1;                                         /* 送信完了フラグの初期化 */
    ……
    CSI00_Start ();                                     /* CSI 通信の開始 */
    CSI00_SendReceiveData ( &txbuf, txnum, &rxbuf );    /* CSI 送受信の開始 */
    while ( gSflag );                                   /* txnum 個の送受信待ち */
    ……
}
```

【CG_serial_user.c】

```
#include    "CG_macrodriver.h"
extern  BOOL    gSflag;                     /* 送信完了フラグ */
__interrupt void MD_INTCSI00 ( void ) {     /* 割り込み INTCSI00 発生時の割り込み処理 */
   if ( gCsi00TxCnt > 0 ) {
        ……
    } else {
        ……
        CSI00_SendEndCallback ();           /* コールバック・ルーチンの呼び出し */
    }
}

void CSI00_SendEndCallback ( void ) {       /* 割り込み INTCSI00 発生時のコールバック・ルーチン */
    gSflag = 0;                             /* 送信完了フラグの設定 */
}
```

CSI*mn*_SendEndCallback

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 MD_INTCSI*mn* のコールバック・ルーチン（CSImn_SendData の引数 *txnum* 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。

[所属]

CG_serial_user.c

[指定形式]

```
void    CSImn_SendEndCallback ( 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

CSI*mn*_ReceiveEndCallback

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 MD_INTCSI*mn* のコールバック・ルーチン（CSImn_ReceiveData の引数 *rxnum* 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。

[所属]

CG_serial_user.c

[指定形式]

```
void    CSImn_ReceiveEndCallback ( 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

CSI*mn*_ErrorCallback

CSI 通信におけるエラー割り込み INTSRE*n* の発生に伴う処理を行います。

備考 本 API 関数は，エラー割り込み INTSRE*n* に対応した割り込み処理 MD_INTSRE*n* のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    CSImn_ErrorCallback ( UCHAR 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	UCHAR *err_type*;	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

[使用例]

以下に，エラー割り込みの発生要因別にコールバック処理を行う際の例を示します。

【CG_serial_user.c】

```
#include    "CG_macrodriver.h"
__interrupt void MD_INTSRE0 ( void ) {        /* 割り込み INTSRE0 発生時の割り込み処理 */
    UCHAR   err_type;
    ……
    CSI00_ErrorCallback ( err_type );         /* コールバック・ルーチンの呼び出し */
}

void CSI00_ErrorCallback ( UCHAR err_type ) { /* 割り込み INTSRE0 発生時のコールバック・ルーチン */
```

```
    if ( err_type & 0x1 ) {                    /* 発生要因の判別 */
        ·····    /* オーバラン・エラーが発生した際のコールバック処理 */
    }
}
```

IIC*mn*_Init

シリアル・インタフェース（簡易 IIC）用チャネルの初期化処理を行います。

備考 本 API 関数は，SAUm_Init 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。

[所属]

CG_serial.c

[指定形式]

```
void    IIC*mn*_Init ( 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

IIC*mn*_Stop

簡易 IIC 通信を終了します。

[所属]

CG_serial.c

[指定形式]

```
void    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

IIC*mn*_MasterSendStart

簡易 IIC マスタ送信を開始します。

備考 本 API 関数では，引数 *txbuf* 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 *txnum* で指定された回数だけ繰り返し行います。

[所属]

CG_serial.c

[指定形式]

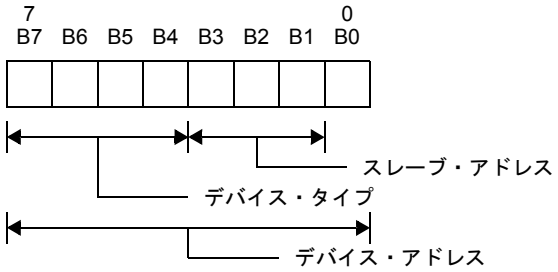
```
#include    "CG_macrodriver.h"
void    IICmn_MasterSendStart ( UCHAR adr, UCHAR *txbuf, USHORT txnum );
```

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	UCHAR *adr*;	デバイス・アドレス
I	UCHAR **txbuf*;	送信するデータを格納したバッファへのポインタ
I	USHORT *txnum*;	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

IIC*mn*_MasterReceiveStart

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 *rxnum* で指定された回数だけ繰り返し行い，引数 *rxbuf* で指定されたバッファに格納します。

[所属]

CG_serial.c

[指定形式]

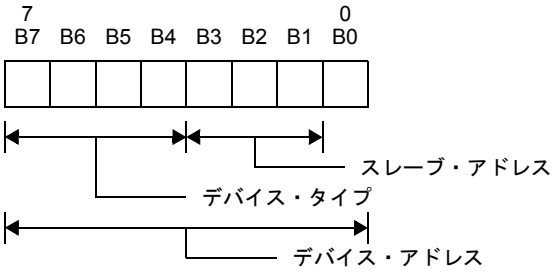
```
#include    "CG_macrodriver.h"
void    IICmn_MasterReceiveStart ( UCHAR adr, UCHAR *rxbuf, USHORT rxnum );
```

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	UCHAR *adr*;	デバイス・アドレス
O	UCHAR **rxbuf*;	受信したデータを格納するバッファへのポインタ
I	USHORT *rxnum*;	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

IIC*mn*_StartCondition

スタート・コンディションを発生します。

備考 本 API 関数は，IICmn_MasterSendStart，および IICmn_MasterReceiveStart 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。

[所属]

CG_serial.c

[指定形式]

```
void    IIC mn_StartCondition ( void );
```

備考 *m* はユニット番号を，*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

IIC*mn*_StopCondition

ストップ・コンディションを発生します。

[所属]

CG_serial.c

[指定形式]

```
void    IIC*mn*_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

IIC*mn*_MasterSendEndCallback

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 MD_INTIIC*mn* のコールバック・ルーチン（IICmn_MasterSendStart の引数 *txnum* 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。

[所属]

CG_serial_user.c

[指定形式]

```
void    IIC*mn*_MasterSendEndCallback ( 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

IIC*mn*_MasterReceiveEndCallback

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 MD_INTIIC*mn* のコールバック・ルーチン（IICmn_MasterReceiveStart の引数 *rxnum* 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。

[所属]

CG_serial_user.c

[指定形式]

```
void    IIC*mn*_MasterReceiveEndCallback ( 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

IIC*mn*_MasterErrorCallback

簡易 IIC 通信におけるパリティ・エラー（ACK エラー）の検出に伴う処理を行います。

[所属]

CG_serial_user.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    IIC*mn*_MasterErrorCallback ( 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

UARTF*n*_Init

シリアル・インタフェース（UARTF*n*）の初期化処理を行います。

[所属]

CG_serial.c

[指定形式]

```
void    UARTFn_Init ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

UARTF*n*_PowerOff

シリアル・インタフェース（UARTF*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，シリアル・インタフェース（UARTF*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（LIN-UART*n* 状態レジスタ：UF*n*STR など）への書き込みは無視されます。

[所属]

CG_serial.c

[指定形式]

```
void    UARTFn_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

UARTF*n*_Start

UARTF 通信を待機状態にします。

[所属]

CG_serial.c

[指定形式]

```
void    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

UARTF*n*_Stop

UARTF 通信を終了します。

[所属]

CG_serial.c

[指定形式]

```
void    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

UARTF*n*_SendData

データの UARTF 送信を開始します。

備考 1. 本 API 関数では，引数 *txbuf* 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 *txnum* 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 UARTFn_Start を呼び出す必要があります。

3. シリアル・インタフェース（UARTF*n*）を拡張ビット・モードで使用する場合，引数 *txbuf* で指定されたバッファには，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”，…

[所属]

CG_serial.c

[指定形式]

```
#include    “CG_macrodriver.h“
MD_STATUS   UARTFn_SendData ( UCHAR *txbuf, USHORT txnum );
```

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	UCHAR *txbuf;	送信するデータを格納したバッファへのポインタ
I	USHORT txnum;	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

UARTF*n*_ReceiveData

データの UARTF 受信を終了にします。

備考 1. 本 API 関数では，1 バイト単位の UARTF 受信を引数 *rxnum* で指定された回数だけ繰り返し行い，引数 *rxbuf* 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，UARTFn_Start を呼び出すことにより開始されます。

3. シリアル・インタフェース（UARTF*n*）を拡張ビット・モードで使用する場合，引数 *rxbuf* で指定されたバッファには，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”，…

[所属]

CG_serial.c

[指定形式]

```
#include    "CG_macrodriver.h"
MD_STATUS   UARTFn_ReceiveData ( UCHAR *rxbuf, USHORT rxnum );
```

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	UCHAR **rxbuf*;	受信したデータを格納するバッファへのポインタ
I	USHORT *rxnum*;	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

UARTF*n*_SetComparisonData

受信データと比較するデータを設定します。

備考 引数 *comdata* に指定された値は，LIN-UART*n* ID 設定レジスタ（UF*n*ID）に設定されます。

[所属]

CG_serial.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    UARTFn_SetComparisonData ( UCHAR comdata );
```

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	UCHAR *comdata*;	比較するデータ

[戻り値]

なし

UARTF*n*_DataComparisonEnable

データの比較を開始します。

備考 本 API 関数を呼び出すことにより，シリアル・インタフェース（UARTF*n*）は，拡張ビット・モード（データ比較あり）へと移行します。

[所属]

CG_serial_user.c

[指定形式]

```
void    UARTFn_DataComparisonEnable ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

UARTF*n*_DataComparisonDisable

データの比較を終了します。

備考 本 API 関数を呼び出すことにより，シリアル・インタフェース（UARTF*n*）は，拡張ビット・モード（データ比較なし）へと移行します。

[所属]

CG_serial_user.c

[指定形式]

```
void    UARTFn_DataComparisonDisable ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

UARTF*n*_SendEndCallback

送信割り込み INTLT*n* の発生に伴う処理を行います。

備考 本 API 関数は，送信割り込み INTLT*n* に対応した割り込み処理 MD_INTLT*n* のコールバック・ルーチン（UARTFn_SendData の引数 *txnum* 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。

[所属]

CG_serial_user.c

[指定形式]

```
void    UARTFn_SendEndCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

UARTF*n*_ReceiveEndCallback

受信完了割り込み INTLR*n* の発生に伴う処理を行います。

備考 本 API 関数は，受信完了割り込み INTLR*n* に対応した割り込み処理 MD_INTLR*n* のコールバック・ルーチン（UARTFn_ReceiveData の引数 *rxnum* 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。

[所属]

CG_serial_user.c

[指定形式]

```
void    UARTFn_ReceiveEndCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

UARTF*n*_SoftOverRunCallback

受信完了割り込み INTLR*n* の発生に伴う処理を行います。

備考 本 API 関数は，受信完了割り込み INTLR*n* に対応した割り込み処理 MD_INTLR*n* のコールバック・ルーチン（UARTFn_ReceiveData の引数 *rxnum* 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。

[所属]

CG_serial_user.c

[指定形式]

```
void    UARTFn_SoftOverRunCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

UARTF*n*_ExpBitCetectCallback

ステータス割り込み INTLS*n* の発生に伴う処理を行います。

備考 本 API 関数は，ステータス割り込み INTLS*n* に対応した割り込み処理 MD_INTLS*n* のコールバック・ルーチン（拡張ビットを受信した際の処理）として呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    UARTFn_ExpBitCetectCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

UARTF*n*_IDMatchCallback

ステータス割り込み INTLS*n* の発生に伴う処理を行います。

備考 本 API 関数は，ステータス割り込み INTLS*n* に対応した割り込み処理 MD_INTLS*n* のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    UARTFn_IDMatchCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

UARTF*n*_ErrorCallback

ステータス割り込み INTLS*n* の発生に伴う処理を行います。

備考 本 API 関数は，ステータス割り込み INTLS*n* に対応した割り込み処理 MD_INTLSn のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    UARTFn_ErrorCallback ( UCHAR 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	UCHAR *err_type*;	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：フレーミング・エラー 000001*xx*B：パリティ・エラー

[戻り値]

なし

IICA_Init

シリアル・インタフェース（IICA）の初期化処理を行います。

[所属]

CG_serial.c

[指定形式]

```
void    IICA_Init ( void );
```

[引数]

なし

[戻り値]

なし

IICA_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，IICA_Init のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    IICA_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

IICA_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。

[所属]

CG_serial.c

[指定形式]

```
void    IICA_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

IICA_Stop

IICA 通信を終了します。

[所属]

CG_serial.c

[指定形式]

```
void    IICA_Stop ( void );
```

[引数]

なし

[戻り値]

なし

IICA_MasterSendStart

IICA マスタ送信を開始します。

備考 本 API 関数では，引数 *txbuf* 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 *txnum* で指定された回数だけ繰り返し行います。

[所属]

CG_serial.c

[指定形式]

```
#include    "CG_macrodriver.h"
MD_STATUS   IICA_MasterSendStart ( UCHAR adr, UCHAR *txbuf, USHORT txnum, UCHAR wait );
```

[引数]

I/O	引数	説明
I	UCHAR *adr*;	スレーブ・アドレス
I	UCHAR **txbuf*;	送信するデータを格納したバッファへのポインタ
I	USHORT *txnum*;	送信するデータの総数
I	UCHAR *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

IICA_MasterReceiveStart

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 *rxnum* で指定された回数だけ繰り返し行い，引数 *rxbuf* で指定されたバッファに格納します。

[所属]

CG_serial.c

[指定形式]

```
#include    "CG_macrodriver.h"
MD_STATUS   IICA_MasterReceiveStart ( UCHAR adr, UCHAR *rxbuf, USHORT rxnum, UCHAR wait );
```

[引数]

I/O	引数	説明
I	UCHAR *adr*;	スレーブ・アドレス
O	UCHAR **rxbuf*;	受信したデータを格納するバッファへのポインタ
I	USHORT *rxnum*;	受信するデータの総数
I	UCHAR *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

IICA_StopCondition

ストップ・コンディションを発生します。

[所属]

CG_serial.c

[指定形式]

```
void    IICA_StopCondition ( void );
```

[引数]

なし

[戻り値]

なし

IICA_MasterSendEndCallback

IICA 通信完了割り込み INTIICA 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 に対応した割り込み処理 MD_INTIICA のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    IICA_MasterSendEndCallback ( void );
```

[引数]

なし

[戻り値]

なし

IICA_MasterReceiveEndCallback

IICA 通信完了割り込み INTIICA 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 に対応した割り込み処理 MD_INTIICA のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    IICA_MasterReceiveEndCallback ( void );
```

[引数]

なし

[戻り値]

なし

IICA_MasterErrorCallback

IICA マスタ通信におけるエラーの検出に伴う処理を行います。

[所属]

CG_serial_user.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    IICA_MasterErrorCallback ( MD_STATUS flag  );
```

[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

IICA_SlaveSendStart

IICA スレーブ送信を開始します。

備考 本 API 関数では，引数 *txbuf* 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 *txnum* で指定された回数だけ繰り返し行います。

[所属]

CG_serial.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    IICA_SlaveSendStart ( UCHAR *txbuf, USHORT txnum );
```

[引数]

I/O	引数	説明
I	UCHAR *txbuf;	送信するデータを格納したバッファへのポインタ
I	USHORT txnum;	送信するデータの総数

[戻り値]

なし

IICA_SlaveReceiveStart

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 *rxnum* で指定された回数だけ繰り返し行い，引数 *rxbuf* で指定されたバッファに格納します。

[所属]

CG_serial.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    IICA_SlaveReceiveStart ( UCHAR *rxbuf, USHORT rxnum );
```

[引数]

I/O	引数	説明
O	UCHAR *rxbuf;	受信したデータを格納するバッファへのポインタ
I	USHORT rxnum;	受信するデータの総数

[戻り値]

なし

IICA_SlaveSendEndCallback

IICA 通信完了割り込み INTIICA 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 に対応した割り込み処理 MD_INTIICA のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    IICA_SlaveSendEndCallback ( void );
```

[引数]

なし

[戻り値]

なし

IICA_SlaveReceiveEndCallback

IICA 通信完了割り込み INTIICA 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 に対応した割り込み処理 MD_INTIICA のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    IICA_SlaveReceiveEndCallback ( void );
```

[引数]

なし

[戻り値]

なし

IICA_SlaveErrorCallback

IICA スレーブ通信におけるエラーの検出に伴う処理を行います。

[所属]

CG_serial_user.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    IICA_SlaveErrorCallback ( MD_STATUS flag  );
```

[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

IICA_GetStopConditionCallback

IICA スレーブ通信におけるストップ・コンディションの検出に伴う処理を行います。

[所属]

CG_serial_user.c

[指定形式]

```
void    IICA_GetStopConditionCallback ( void );
```

[引数]

なし

[戻り値]

なし

IIC*n*_Init

シリアル・インタフェース（IIC*n*）の初期化処理を行います。

[所属]

CG_serial.c

[指定形式]

```
void    IICn_Init ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

IICn_UserInit

シリアル・インタフェース（IICn）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，IICn_Init のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    IICn_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

IIC*n*_Stop

IIC*n* 通信を終了します。

[所属]

CG_serial.c

[指定形式]

```
void    IICn_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

IIC*n*_MasterSendStart

IIC*n* マスタ送信を開始します。

備考 本 API 関数では，引数 *txbuf* で指定されたバッファから 1 バイト単位の IIC*n* マスタ送信を引数 *txnum* で指定された回数だけ繰り返し行います。

[所属]

CG_serial.c

[指定形式]

```
#include    "CG_macrodriver.h"
MD_STATUS   IICn_MasterSendStart ( UCHAR adr, UCHAR *txbuf, USHORT txnum, UCHAR 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	UCHAR *adr*;	スレーブ・アドレス
I	UCHAR **txbuf*;	送信するデータを格納したバッファへのポインタ
I	USHORT *txnum*;	送信するデータの総数
I	UCHAR *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_ERROR	異常終了

IIC*n*_MasterReceiveStart

IIC*n* 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 IIC*n* マスタ受信を引数 *rxnum* で指定された回数だけ繰り返し行い，引数 *rxbuf* で指定されたバッファに格納します。

[所属]

CG_serial.c

[指定形式]

```
#include    "CG_macrodriver.h"
MD_STATUS   IICn_MasterReceiveStart ( UCHAR adr, UCHAR *rxbuf, USHORT rxnum, UCHAR 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	UCHAR *adr*;	スレーブ・アドレス
O	UCHAR **rxbuf*;	受信したデータを格納するバッファへのポインタ
I	USHORT *rxnum*;	受信するデータの総数
I	UCHAR *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_ERROR	異常終了

IIC*n*_MasterSendEndCallback

IIC*n* 通信完了割り込み INTIIC*n* の発生に伴う処理を行います。

備考 本 API 関数は，IIC*n* 通信完了割り込み INTIIC*n* に対応した割り込み処理 MD_INTIIC*n* のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    IICn_MasterSendEndCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

IIC*n*_MasterReceiveEndCallback

IIC*n* 通信完了割り込み INTIIC*n* の発生に伴う処理を行います。

備考 本 API 関数は，IIC*n* 通信完了割り込み INTIIC*n* に対応した割り込み処理 MD_INTIIC*n* のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    IICn_MasterReceiveEndCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

IIC*n*_MasterErrorCallback

IIC*n* マスタ通信におけるエラーの検出に伴う処理を行います。

[所属]

CG_serial_user.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    IICn_MasterErrorCallback ( 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

IIC*n*_SlaveSendStart

IIC*n* スレーブ送信を開始します。

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

[所属]

CG_serial.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    IICn_SlaveSendStart ( UCHAR *txbuf, USHORT txnum );
```

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	UCHAR *txbuf;	送信するデータを格納したバッファへのポインタ
I	USHORT txnum;	送信するデータの総数

[戻り値]

なし

IIC*n*_SlaveReceiveStart

IIC*n* 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 IIC*n* スレーブ受信を引数 *rxnum* で指定された回数だけ繰り返し行い，引数 *rxbuf* で指定されたバッファに格納します。

[所属]

CG_serial.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    IICn_SlaveReceiveStart ( UCHAR *rxbuf, USHORT rxnum );
```

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	UCHAR *rxbuf;	受信したデータを格納するバッファへのポインタ
I	USHORT rxnum;	受信するデータの総数

[戻り値]

なし

IIC*n*_SlaveSendEndCallback

IIC*n* 通信完了割り込み INTIIC*n* の発生に伴う処理を行います。

備考 本 API 関数は，IIC*n* 通信完了割り込み INTIIC*n* に対応した割り込み処理 MD_INTIIC*n* のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    IICn_SlaveSendEndCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

IIC*n*_SlaveReceiveEndCallback

IIC*n* 通信完了割り込み INTIIC*n* の発生に伴う処理を行います。

備考 本 API 関数は，IIC*n* 通信完了割り込み INTIIC*n* に対応した割り込み処理 MD_INTIIC*n* のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    IICn_SlaveReceiveEndCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

IIC*n*_SlaveErrorCallback

IIC*n* スレーブ通信におけるエラーの検出に伴う処理を行います。

[所属]

CG_serial_user.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    IICn_SlaveErrorCallback ( 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

IIC*n*_GetStopConditionCallback

IIC*n* スレーブ通信におけるストップ・コンディションの検出に伴う処理を行います。

[所属]

CG_serial_user.c

[指定形式]

```
void    IICn_GetStopConditionCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

C. 3. 6 オペアンプ

以下に，コード生成がオペアンプ用として出力する API 関数の一覧を示します。

表 C—7 オペアンプ用 API 関数

API 関数名	機能概要
OPAMP_Init	オペアンプの機能を制御するうえで必要となる初期化処理を行います。
OPAMP_UserInit	オペアンプに関するユーザ独自の初期化処理を行います。
AMPn_Start	オペアンプ *n*（シングル・アンプ・モード）の動作を開始します。
AMPn_Stop	オペアンプ *n*（シングル・アンプ・モード）の動作を停止します。

OPAMP_Init

オペアンプの機能を制御するうえで必要となる初期化処理を行います。

[所属]

CG_opamp.c

[指定形式]

```
void    OPAMP_Init ( void );
```

[引数]

なし

[戻り値]

なし

OPAMP_UserInit

オペアンプに関するユーザ独自の初期化処理を行います。

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

[所属]

CG_opamp_user.c

[指定形式]

```
void    OPAMP_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

AMP*n*_Start

オペアンプ *n*（シングル・アンプ・モード）の動作を開始します。

[所属]

CG_opamp.c

[指定形式]

```
void    AMPn_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

AMP*n*_Stop

オペアンプ *n*（シングル・アンプ・モード）の動作を停止します。

[所属]

CG_opamp.c

[指定形式]

```
void    AMPn_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

C. 3. 7 コンパレータ／PG アンプ

以下に，コード生成がコンパレータ／PG アンプ用として出力する API 関数の一覧を示します。

表 C—8 コンパレータ／PG アンプ用 API 関数

API 関数名	機能概要
CMPPGA_Init	コンパレータ／プログラマブル・ゲイン・アンプの機能を制御するうえで必要となる初期化処理を行います。
CMPPGA_UserInit	コンパレータ／プログラマブル・ゲイン・アンプに関するユーザ独自の初期化処理を行います。
CMPPGA_PowerOff	コンパレータ／プログラマブル・ゲイン・アンプに対するクロック供給を停止します。
CMPPGA_Start	コンパレータ／プログラマブル・ゲイン・アンプの動作を開始します。
CMPPGA_Stop	コンパレータ／プログラマブル・ゲイン・アンプの動作を停止します。
CMPPGA_ChangeCMPnRefVoltage	コンパレータ *n* の内蔵基準電圧を設定します。
CMPPGA_ChangePGAFactor	プログラマブル・ゲイン・アンプにおける入力電圧の増幅率を設定します。

CMPPGA_Init

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

[所属]

CG_cmppga.c

[指定形式]

```
void    CMPPGA_Init ( void );
```

[引数]

なし

[戻り値]

なし

CMPPGA_UserInit

コンパレータ／プログラマブル・ゲイン・アンプに関するユーザ独自の初期化処理を行います。

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

[所属]

CG_cmppga_user.c

[指定形式]

```
void    CMPPGA_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

CMPPGA_PowerOff

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

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

[所属]

CG_cmppga.c

[指定形式]

```
void    CMPPGA_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

CMPPGA_Start

コンパレータ／プログラマブル・ゲイン・アンプの動作を開始します。

[所属]

CG_cmppga.c

[指定形式]

```
void    CMPPGA_Start ( void );
```

[引数]

なし

[戻り値]

なし

CMPPGA_Stop

コンパレータ／プログラマブル・ゲイン・アンプの動作を停止します。

[所属]

CG_cmppga.c

[指定形式]

```
void    CMPPGA_Stop ( void );
```

[引数]

なし

[戻り値]

なし

CMPPGA_ChangeCMP*n*RefVoltage

コンパレータ *n* の内蔵基準電圧を設定します。

備考 引数 *voltage* に指定された値は，コンパレータ *n* 内蔵基準電圧選択レジスタ（C*n*RVM）に設定されます。

[所属]

CG_cmppga.c

[指定形式]

```
#include    "CG_macrodriver.h"
#include    "CG_cmppga.h"
MD_STATUS   CMPPGA_ChangeCMPnRefVoltage ( enum CMPRefVoltage voltage );
```

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	enum CMPRefVoltage *voltage*;	コンパレータ *n* の内蔵基準電圧 【*n* = 0：チャネル 0 の場合】 CMPREFVOL0： 2AVREF/16 CMPREFVOL1： 4AVREF/16 CMPREFVOL2： 6AVREF/16 CMPREFVOL3： 8AVREF/16 CMPREFVOL4： 10AVREF/16 CMPREFVOL5： 12AVREF/16 【*n* = 1：チャネル 1 の場合】 CMPREFVOL0： 3AVREF/16 CMPREFVOL1： 5AVREF/16 CMPREFVOL2： 7AVREF/16 CMPREFVOL3： 9AVREF/16 CMPREFVOL4： 11AVREF/16 CMPREFVOL5： 13AVREF/16

[戻り値]

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

CMPPGA_ChangePGAFactor

プログラマブル・ゲイン・アンプにおける入力電圧の増幅率を設定します。

備考 引数 *factor* に指定された値は，プログラマブル・ゲイン・アンプ制御レジスタ（OAM）に設定されます。

[所属]

CG_cmppga.c

[指定形式]

```
#include    "CG_macrodriver.h"
#include    "CG_cmppga.h"
MD_STATUS   CMPPGA_ChangePGAFactor ( enum PGAFactor factor );
```

[引数]

I/O	引数	説明
I	enum PGAFactor *factor*;	入力電圧の増幅率 PGAFACTOR0： 4 倍 PGAFACTOR1： 6 倍 PGAFACTOR2： 8 倍 PGAFACTOR3： 10 倍 PGAFACTOR4： 12 倍

[戻り値]

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

C. 3. 8 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—9 A/D コンバータ用 API 関数

API 関数名	機能概要
AD_Init	A/D コンバータの機能を制御するうえで必要となる初期化処理を行います。
AD_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。
AD_PowerOff	A/D コンバータに対するクロック供給を停止します。
AD_ComparatorOn	電圧コンパレータを動作許可状態に設定します。
AD_ComparatorOff	電圧コンパレータを動作停止状態に設定します。
AD_Start	A/D 変換を開始します。
AD_Stop	A/D 変換を終了します。
AD_SelectADChannel	A/D 変換するアナログ電圧の入力端子を設定します。
AD_Read	A/D 変換結果を読み出します。
AD_ReadByte	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。

AD_Init

A/D コンバータの機能を制御するうえで必要となる初期化処理を行います。

[所属]

CG_ad.c

[指定形式]

```
void    AD_Init ( void );
```

[引数]

なし

[戻り値]

なし

AD_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，AD_Init のコールバック・ルーチンとして呼び出されます。

[所属]

CG_ad_user.c

[指定形式]

```
void    AD_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

AD_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（A/D コンバータ・モード・レジスタ：ADCM など）への書き込みは無視されます。

[所属]

CG_ad.c

[指定形式]

```
void    AD_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

AD_ComparatorOn

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 AD_Start の間には，約 1μ 秒の時間を空ける必要があります。

2. コード生成 パネル（[A/D コンバータ]）の［コンパレータの動作設定］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。

[所属]

CG_ad.c

[指定形式]

```
void    AD_ComparatorOn ( void );
```

[引数]

なし

[戻り値]

なし

AD_ComparatorOff

電圧コンパレータを動作停止状態に設定します。

[所属]

CG_ad.c

[指定形式]

```
void    AD_ComparatorOff ( void );
```

[引数]

なし

[戻り値]

なし

AD_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，AD_ComparatorOn 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。

[所属]

CG_ad.c

[指定形式]

```
void    AD_Start ( void );
```

[引数]

なし

[戻り値]

なし

[使用例 1]

以下に，コード生成 パネル（[A/D コンバータ]）で選択された変換開始端子からのアナログ電圧を A/D 変換したのち，入力端子 ANI1 からのアナログ電圧を A/D 変換する際の例を示します。

なお，下記は，コード生成 パネル（[A/D コンバータ]）の［コンパレータの動作設定］エリアで，“停止”が選択された場合（AD_ComparatorOn の呼び出しを行う場合）の例となっています。

【CG_main.c】

```
#include    "CG_macrodriver.h"
#include    "CG_ad.h"
BOOL    gFlag;                          /* A/D 変換完了フラグ */
void main ( void ) {
    USHORT  buffer = 0;
    int     wait = 100;
    gFlag = 1;                          /* A/D 変換完了フラグの初期化 */
    ……
    AD_ComparatorOn ();                 /* 動作許可状態への移行 */
    while ( wait );                     /* 安定時間の確保（1μ 秒以上） */
    AD_Start ();                        /* A/D 変換の開始 */
```

```
    while ( gFlag );                    /* 割り込み INTAD の発生待ち */
    AD_Read ( &buffer );                /* A/D 変換結果の読み出し */
    AD_SelectADChannel ( ADCHANNEL1 );  /* 入力端子の切り替え */
    gFlag = 1;                          /* A/D 変換完了フラグの初期化 */
    while ( gFlag );                    /* 割り込み INTAD の発生待ち */
    AD_Read ( &buffer );                /* A/D 変換結果の読み出し */
    AD_Stop ();                         /* A/D 変換の終了 */
    AD_ComparatorOff ();                /* 動作停止状態への移行 */
    ……
}
```

【CG_ad_user.c】

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                  /* A/D 変換完了フラグ */
__interrupt void MD_INTAD ( void ) {    /* 割り込み INTAD 発生時の割り込み処理 */
    gFlag = 0;                          /* A/D 変換完了フラグの設定 */
}
```

[使用例 2]

以下に，コード生成 パネル（[A/D コンバータ]）で選択された変換開始端子からのアナログ電圧を A/D 変換したのち，入力端子 ANI1 からのアナログ電圧を A/D 変換する際の例を示します。

なお，下記は，コード生成 パネル（[A/D コンバータ]）の［コンパレータの動作設定］エリアで，“許可”が選択された場合（AD_ComparatorOn の呼び出しを行わない場合）の例となっています。

【CG_main.c】

```
#include    "CG_macrodriver.h"
#include    "CG_ad.h"
BOOL    gFlag;                          /* A/D 変換完了フラグ */
void main ( void ) {
    USHORT  buffer = 0;
    gFlag = 1;                          /* A/D 変換完了フラグの初期化 */
    ……
    AD_Start ();                        /* A/D 変換の開始 */
    while ( gFlag );                    /* 割り込み INTAD の発生待ち */
    AD_Read ( &buffer );                /* A/D 変換結果の読み出し */
    AD_SelectADChannel ( ADCHANNEL1 );  /* 入力端子の切り替え */
    gFlag = 1;                          /* A/D 変換完了フラグの初期化 */
    while ( gFlag );                    /* 割り込み INTAD の発生待ち */
    AD_Read ( &buffer );                /* A/D 変換結果の読み出し */
    AD_Stop ();                         /* A/D 変換の終了 */
    ……
}
```

【CG_ad_user.c】

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                  /* A/D 変換完了フラグ */
__interrupt void MD_INTAD ( void ) {    /* 割り込み INTAD 発生時の割り込み処理 */
    gFlag = 0;                          /* A/D 変換完了フラグの設定 */
}
```

AD_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，AD_ComparatorOff を呼び出す必要があります。

[所属]

CG_ad.c

[指定形式]

```
void    AD_Stop ( void );
```

[引数]

なし

[戻り値]

なし

AD_SelectADChannel

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。

[所属]

CG_ad.c

[指定形式]

- 【Fx3】【Ix3】【Kx3-A】【Kx3-L】【Lx3】

```
#include    "CG_ad.h"
MD_STATUS   AD_SelectADChannel ( enum ADChannel channel );
```

- 【Kx3】

```
#include    "CG_ad.h"
void    AD_SelectADChannel ( enum ADChannel channel );
```

[引数]

I/O	引数	説明
I	enum ADChannel *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 CG_ad.h を参照してください。

[戻り値]

- 【Fx3】【Ix3】【Kx3-A】【Kx3-L】【Lx3】

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

- 【Kx3】

なし

AD_Read

A/D 変換結果を読み出します。

備考 読み出す A/D 変換結果は，対象デバイスが 78K0R/Fx3，78K0R/Kx3，78K0R/Kx3-L の場合は 10 ビット，78K0R/Kx3-A，78K0R/Lx3 の場合は 12 ビットとなります。

[所属]

CG_ad.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    AD_Read ( USHORT *buffer );
```

[引数]

I/O	引数	説明
O	USHORT *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

AD_ReadByte

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。

[所属]

CG_ad.c

[指定形式]

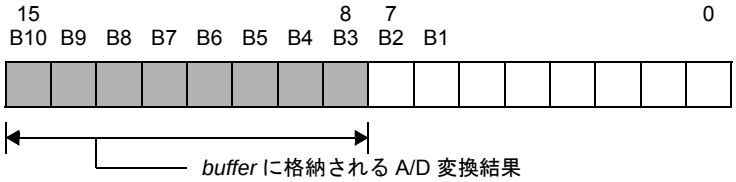
```
#include    "CG_macrodriver.h"
void    AD_ReadByte ( UCHAR *buffer );
```

[引数]

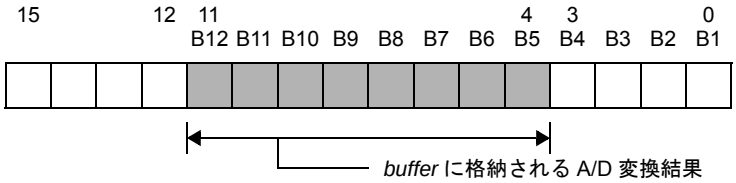
I/O	引数	説明
O	UCHAR *buffer;	読み出した A/D 変換結果（8 ビット：10 ビット分解能の上位 8 ビット）を格納する領域へのポインタ

備考 以下に，*buffer* に格納される A/D 変換結果を示します。

- 【Fx3】【Ix3】【Kx3】【Kx3-L】



- 【Kx3-A】【Lx3】



[戻り値]

なし

C. 3. 9 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—10 D/A コンバータ用 API 関数

API 関数名	機能概要
DA_Init	D/A コンバータの機能を制御するうえで必要となる初期化処理を行います。
DA_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。
DA_PowerOff	D/A コンバータに対するクロック供給を停止します。
DAn_Start	D/A 変換を開始します。
DAn_Stop	D/A 変換を終了します。
DAn_SetValue	ANO*n* 端子に出力するアナログ電圧値を設定します。
DAn_Set8BitsValue	ANO*n* 端子に出力するアナログ電圧値（8 ビット）を設定します。
DAn_Set12BitsValue	ANO*n* 端子に出力するアナログ電圧値（12 ビット）を設定します。

DA_Init

D/A コンバータの機能を制御するうえで必要となる初期化処理を行います。

[所属]

CG_da.c

[指定形式]

```
void    DA_Init ( void );
```

[引数]

なし

[戻り値]

なし

DA_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，DA_Init のコールバック・ルーチンとして呼び出されます。

[所属]

CG_da_user.c

[指定形式]

```
void    DA_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

DA_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（D/A コンバータ・モード・レジスタ：DAM など）への書き込みは無視されます。

[所属]

CG_da.c

[指定形式]

```
void    DA_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

DA*n*_Start

D/A 変換を開始します。

[所属]

CG_da.c

[指定形式]

```
void    DAn_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

DA*n*_Stop

D/A 変換を終了します。

[所属]

CG_da.c

[指定形式]

```
void    DAn_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

DA*n*_SetValue

ANO*n* 端子に出力するアナログ電圧値を設定します。

[所属]

CG_da.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    DAn_SetValue ( UCHAR 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	UCHAR *value*;	アナログ電圧値（0x0 ～ 0xff）

[戻り値]

なし

[使用例]

以下に，チャネル 0，およびチャネル 1 に“アナログ電圧値”を設定する際の例を示します。

【CG_main.c】

```
void main ( void ) {
    .........
    DA0_Start ();                       /* D/A 変換の開始 */
    DA1_Start ();                       /* D/A 変換の開始 */
    .........
    DA0_SetValue ( 0x7f );              /* アナログ電圧値の設定 */
    .........
}
```

【CG_timer_user.c】

```
#include    "CG_macrodriver.h"
```

```
UCHAR   gValue = 0;
__interrupt void MD_INTTM05 ( void ) {  /* 割り込み INTTM05 発生時の割り込み処理 */
    DA1_SetValue ( gValue++ );          /* アナログ電圧値の設定 */
}
```

DA*n*_Set8BitsValue

ANO*n* 端子に出力するアナログ電圧値（8 ビット）を設定します。

[所属]

CG_da.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    DAn_Set8BitsValue ( UCHAR 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	UCHAR *value*;	アナログ電圧値（0x0 ～ 0x1f）

[戻り値]

なし

[使用例]

以下に，チャネル 0，およびチャネル 1 に“アナログ電圧値”を設定する際の例を示します。

【CG_main.c】

```
void main ( void ) {
    .........
    DA0_Start ();                       /* D/A 変換の開始 */
    DA1_Start ();                       /* D/A 変換の開始 */
    .........
    DA0_Set8BitsValue ( 0x7f );         /* アナログ電圧値の設定 */
    .........
}
```

【CG_timer_user.c】

```
#include    "CG_macrodriver.h"
```

```
UCHAR   gValue = 0;
__interrupt void MD_INTTM05 ( void ) {  /* 割り込み INTTM05 発生時の割り込み処理 */
    DA1_Set8BitsValue ( gValue++ );     /* アナログ電圧値の設定 */
}
```

DA*n*_Set12BitsValue

ANO*n* 端子に出力するアナログ電圧値（12 ビット）を設定します。

[所属]

CG_da.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    DAn_Set12BitsValue ( UCHAR 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	UCHAR *value*;	アナログ電圧値（0x0 ～ 0xfff）

[戻り値]

なし

[使用例]

以下に，チャネル 0，およびチャネル 1 に“アナログ電圧値”を設定する際の例を示します。

【CG_main.c】

```
void main ( void ) {
    .........
    DA0_Start ();                       /* D/A 変換の開始 */
    DA1_Start ();                       /* D/A 変換の開始 */
    .........
    DA0_Set12BitsValue ( 0x1ff );       /* アナログ電圧値の設定 */
    .........
}
```

【CG_timer_user.c】

```
#include    "CG_macrodriver.h"
```

```
UCHAR   gValue = 0;
__interrupt void MD_INTTM05 ( void ) {  /* 割り込み INTTM05 発生時の割り込み処理 */
    DA1_Set12BitsValue ( gValue++ );    /* アナログ電圧値の設定 */
}
```

C. 3. 10 タ イ マ

以下に，コード生成がタイマ用として出力する API 関数の一覧を示します。

表 C—11 タイマ用 API 関数

API 関数名	機能概要
TAUm_Init	タイマ・アレイ・ユニットの機能を制御するうえで必要となる初期化処理を行います。
TAUm_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。
TAUm_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。
TAUm_ChannelnStart	チャネル *n* のカウントを開始します。
TAUm_ChannelnStop	チャネル *n* のカウントを終了します。
TAUm_ChannelnChangeCondition	カウント値を変更します。
TAUm_ChannelnChangeTimerCondition	カウント値を変更します。
TAUm_ChannelnGetPulseWidth	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。
TAUm_ChannelnChangeDuty	TO*mn* 端子に出力する PWM 信号のデューティ比を変更します。
TAUm_ChannelnSoftWareTriggerOn	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。

TAU*m*_Init

タイマ・アレイ・ユニットの機能を制御するうえで必要となる初期化処理を行います。

[所属]

CG_timer.c

[指定形式]

```
void    TAUm_Init ( 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

TAU*m*_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，TAUm_Init のコールバック・ルーチンとして呼び出されます。

[所属]

CG_timer_user.c

[指定形式]

```
void    TAUm_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

TAU*m*_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（タイマ・クロック選択レジスタ 0：TPS0 など）への書き込みは無視されます。

[所属]

CG_timer.c

[指定形式]

```
void    TAUm_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

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。

[所属]

CG_timer.c

[指定形式]

```
void    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

TAU*m*_Channel*n*_Stop

チャネル *n* のカウントを終了します。

[所属]

CG_timer.c

[指定形式]

```
void    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

TAU*m*_Channel*n*_ChangeCondition

カウント値を変更します。

備考 1. 引数 *regvalue* に指定された値は，タイマ・データ・レジスタ *mn*（TDR*mn*）に設定されます。

2. 本 API 関数の呼び出しタイミングは，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）により以下のように異なります。

機能の種類	呼び出しタイミング
インターバル・タイマ	任意のタイミングで可
方形波出力	任意のタイミングで可
分周器機能	任意のタイミングで可
外部イベント・カウンタ	任意のタイミングで可
入力パルス間隔測定	呼び出し不可
入力信号のハイ／ロウ・レベル幅測定	呼び出し不可
PWM 出力	呼び出し不可
ワンショット・パルス出力	動作中は呼び出し不可
多重 PWM 出力	呼び出し不可

[所属]

CG_timer.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    TAUm_Channeln_ChangeCondition ( USHORT regvalue );
```

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	USHORT *regvalue*;	カウント値（0x0 ～ 0xffff）

[戻り値]

なし

［使用例］

以下に，インターバル時間を半分に変更する際の例を示します。

なお，下記は，チャネル 0 がインターバル・タイマ用として選択された場合の例となっています。

【CG_main.c】

```
#include    "CG_macrodriver.h"
void main ( void ) {
    USHORT  value = TAU_TDR00_VALUE >> 1;       /* TAU_TDR00_VALUE：現在のインターバル時間 */
    .........
    TAU0_Channel0_Start ();                     /* カウントの開始 */
    .........
    TAU0_Channel0_ChangeCondition ( value );    /* カウント値の変更 */
   .........
}
```

TAU*m*_Channel*n*_ChangeTimerCondition

カウント値を変更します。

備考 1. 引数 *regvalue* に指定された値は，タイマ・データ・レジスタ *mn*（TDR*mn*）に設定されます。

2. 本 API 関数の呼び出しタイミングは，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）により以下のように異なります。

機能の種類	呼び出しタイミング
インターバル・タイマ	任意のタイミングで可
方形波出力	任意のタイミングで可
分周器機能	任意のタイミングで可
外部イベント・カウンタ	任意のタイミングで可
入力パルス間隔測定	呼び出し不可
入力信号のハイ／ロウ・レベル幅測定	呼び出し不可
PWM 出力	呼び出し不可
ワンショット・パルス出力	動作中は呼び出し不可
多重 PWM 出力	呼び出し不可

[所属]

CG_timer.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    TAUm_Channeln_ChangeTimerCondition ( USHORT regvalue );
```

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	USHORT *regvalue*;	カウント値（0x0 ～ 0xffff）

[戻り値]

なし

［使用例］

以下に，インターバル時間を半分に変更する際の例を示します。

なお，下記は，チャネル 0 がインターバル・タイマ用として選択された場合の例となっています。

【CG_main.c】

```
#include    "CG_macrodriver.h"
void main ( void ) {
    USHORT  value = TAU_TDR00_VALUE >> 1;          /* TAU_TDR00_VALUE：現在のインターバル時間 */
    .........
    TAU0_Channel0_Start ();                        /* カウントの開始 */
    .........
    TAU0_Channel0_ChangeTimerCondition ( value ); /* カウント値の変更 */
   .........
}
```

TAU*m*_Channel*n*_GetPulseWidth

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。

[所属]

CG_timer.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    TAUm_Channeln_GetPulseWidth ( ULONG *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	ULONG *width;	測定幅（0x0 ～ 0x1ffff）を格納する領域へのポインタ

[戻り値]

なし

TAU*m*_Channel*n*_ChangeDuty

TO*mn* 端子に出力する PWM 信号のデューティ比を変更します。

備考 本 API 関数の呼び出しタイミングは，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）により以下のように異なります。

機能の種類	呼び出しタイミング
インターバル・タイマ	呼び出し不可
方形波出力	呼び出し不可
分周器機能	呼び出し不可
外部イベント・カウンタ	呼び出し不可
入力パルス間隔測定	呼び出し不可
入力信号のハイ／ロウ・レベル幅測定	呼び出し不可
PWM 出力	マスタ・チャネルで割り込み INTTM*mn* が発生した後
ワンショット・パルス出力	呼び出し不可
多重 PWM 出力	マスタ・チャネルで割り込み INTTM*mn* が発生した後

[所属]

CG_timer.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    TAUm_Channeln_ChangeDuty ( UCHAR ratio );
```

備考 *m* はユニット番号を，*n* はスレーブ側のチャネル番号を意味します。

[引数]

I/O	引数	説明
I	UCHAR *ratio*;	デューティ比（0 ～ 100，単位：%）

備考 デューティ比 *ratio* に設定する値は，10 進数に限られます。

[戻り値]

なし

［使用例］

以下に，デューティ比を 25% に変更する際の例を示します。

なお，下記は，チャネル 0，1 が PWM 出力，または多重 PWM 出力用として選択された場合の例となっています。

【CG_main.c】

```
#include    "CG_macrodriver.h"
void main ( void ) {
    UCHAR   ratio = 25;
    .........
    TAU0_Channel0_Start ();             /* カウントの開始 */
    .........
    TAU0_Channel1_ChangeDuty ( ratio ); /* デューティ比の変更 */
    .........
}
```

TAU*m*_Channel*n*_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。

[所属]

CG_timer.c

[指定形式]

```
void    TAUm_Channeln_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. 3. 11 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—12 ウォッチドッグ・タイマ用 API 関数

API 関数名	機能概要
WDT_Init	ウォッチドッグ・タイマの機能を制御するうえで必要となる初期化処理を行います。
WDT_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。
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。

WDT_Init

ウォッチドッグ・タイマの機能を制御するうえで必要となる初期化処理を行います。

[所属]

CG_wdt.c

[指定形式]

```
void    WDT_Init ( void );
```

[引数]

なし

[戻り値]

なし

WDT_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，WDT_Init のコールバック・ルーチンとして呼び出されます。

[所属]

CG_wdt_user.c

[指定形式]

```
void    WDT_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

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。

[所属]

CG_wdt.c

[指定形式]

```
void    WDT_Restart ( void );
```

[引数]

なし

[戻り値]

なし

C. 3. 12 リアルタイム・カウンタ

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

表 C—13 リアルタイム・カウンタ用 API 関数

API 関数名	機能概要
RTC_Init	リアルタイム・カウンタの機能を制御するうえで必要となる初期化処理を行います。
RTC_UserInit	リアルタイム・カウンタに関するユーザ独自の初期化処理を行います。
RTC_PowerOff	リアルタイム・カウンタに対するクロック供給を停止します。
RTC_CounterEnable	リアルタイム・カウンタ（年，月，曜日，日，時，分，秒）のカウントを開始します。
RTC_CounterDisable	リアルタイム・カウンタ（年，月，曜日，日，時，分，秒）のカウントを終了します。
RTC_SetHourSystem	リアルタイム・カウンタの時間制（12 時間制，24 時間制）を設定します。
RTC_CounterSet	リアルタイム・カウンタにカウント値（年，月，曜日，日，時，分，秒）を設定します。
RTC_CounterGet	リアルタイム・カウンタのカウント値（年，月，曜日，日，時，分，秒）を読み出します。
RTC_ConstPeriodInterruptEnable	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。
RTC_ConstPeriodInterruptDisable	定周期割り込み機能を終了します。
RTC_ConstPeriodInterruptCallback	定周期割り込み INTRTC の発生に伴う処理を行います。
RTC_AlarmEnable	アラーム割り込み機能を開始します。
RTC_AlarmDisable	アラーム割り込み機能を終了します。
RTC_AlarmSet	アラームの発生条件（曜日，時，分）を設定します。
RTC_AlarmGet	アラームの発生条件（曜日，時，分）を読み出します。
RTC_AlarmInterruptCallback	アラーム割り込み INTRTC の発生に伴う処理を行います。
RTC_IntervalStart	インターバル割り込み機能を開始します。
RTC_IntervalStop	インターバル割り込み機能を終了します。
RTC_IntervalInterruptEnable	割り込み INTRTCI の発生周期を設定したのち，インターバル割り込み機能を開始します。
RTC_IntervalInterruptDisable	インターバル割り込み機能を終了します。
RTC_RTC1HZ_OutputEnable	RTC1HZ 端子に対するリアルタイム・カウンタ補正クロック（1 Hz）の出力を許可します。
RTC_RTC1HZ_OutputDisable	RTC1HZ 端子に対するリアルタイム・カウンタ補正クロック（1 Hz）の出力を禁止します。
RTC_RTCCL_OutputEnable	RTCCL 端子に対するリアルタイム・カウンタ・クロック（32 kHz 原発）の出力を許可します。
RTC_RTCCL_OutputDisable	RTCCL 端子に対するリアルタイム・カウンタ・クロック（32 kHz 原発）の出力を禁止します。

API 関数名	機能概要
RTC_RTCDIV_OutputEnable	RTCDIV 端子に対するリアルタイム・カウンタ・クロック（32 kHz 分周）の出力を許可します。
RTC_RTCDIV_OutputDisable	RTCDIV 端子に対するリアルタイム・カウンタ・クロック（32 kHz 分周）の出力を禁止します。
RTC_ChangeCorrectionValue	時計誤差を補正するタイミング，および補正値を変更します。

RTC_Init

リアルタイム・カウンタの機能を制御するうえで必要となる初期化処理を行います。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_Init ( void );
```

[引数]

なし

[戻り値]

なし

RTC_UserInit

リアルタイム・カウンタに関するユーザ独自の初期化処理を行います。

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

[所属]

CG_rtc_user.c

[指定形式]

```
void    RTC_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

RTC_PowerOff

リアルタイム・カウン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，リアルタイム・カウン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（リアルタイム・カウンタ・コントロール・レジスタ 0：RTCC0 など）への書き込みは無視されます。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

RTC_CounterEnable

リアルタイム・カウンタ（年，月，曜日，日，時，分，秒）のカウントを開始します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_CounterEnable ( void );
```

[引数]

なし

[戻り値]

なし

RTC_CounterDisable

リアルタイム・カウンタ（年，月，曜日，日，時，分，秒）のカウントを終了します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_CounterDisable ( void );
```

[引数]

なし

[戻り値]

なし

RTC_SetHourSystem

リアルタイム・カウンタの時間制（12 時間制，24 時間制）を設定します。

[所属]

CG_rtc.c

[指定形式]

```
#include    "CG_macrodriver.h"
#include    "CG_rtc.h"
MD_STATUS   RTC_SetHourSystem ( enum RTCHourSystem hoursystem );
```

[引数]

I/O	引数	説明
I	enum RTCHourSystem *hoursystem*;	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 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。

[使用例]

以下に，リアルタイム・カウンタの時間制を“24 時間制”に設定する際の例を示します。

【CG_main.c】

```
#include    "CG_rtc.h"
void main ( void ) {
```

```
    .........
    RTC_CounterEnable ();           /* カウントの開始 */
    .........
    RTC_SetHourSystem ( HOUR24 );   /* 時間制の設定 */
    .........
}
```

RTC_CounterSet

リアルタイム・カウンタにカウント値を設定します。

[所属]

CG_rtc.c

[指定形式]

```
#include    "CG_macrodriver.h"
#include    "CG_rtc.h"
MD_STATUS   RTC_CounterSet ( struct RTCCounterValue counterwriteval );
```

[引数]

I/O	引数	説明
I	struct RTCCounterValue *counterwriteval*;	カウント値

備考 以下に，リアルタイム・カウンタのカウント値 RTCCounterValue の構成を示します。

```
struct  RTCCounterValue {
    UCHAR   Sec;    /* 秒 */
    UCHAR   Min;    /* 分 */
    UCHAR   Hour;   /* 時 */
    UCHAR   Day;    /* 日 */
    UCHAR   Week;   /* 曜日（0：日曜日，6：土曜日）*/
    UCHAR   Month;  /* 月 */
    UCHAR   Year;   /* 年 */
};
```

[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 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。

［使用例］

以下に，リアルタイム・カウンタのカウント値として，“2008 年 12 月 25 日（木）17 時 30 分 00 秒”を設定する際の例を示します。

【CG_main.c】

```
#include    "CG_rtc.h"
void main ( main ) {
    struct  RTCCounterValue counterwriteval;
    ……
    RTC_CounterEnable ();               /* カウントの開始 */
    ……
    counterwriteval.Year = 0x08;
    counterwriteval.Month = 0x12;
    counterwriteval.Day = 0x25;
    counterwriteval.Week = 0x05;
    counterwriteval.Hour = 0x17;
    counterwriteval.Min = 0x30;
    counterwriteval.Sec = 0;
    RTC_SetHourSystem ( HOUR24 );       /* 時間制の設定 */
    RTC_CounterSet ( counterwriteval ); /* カウント値の設定 */
    ……
}
```

RTC_CounterGet

リアルタイム・カウンタのカウント値を読み出します。

[所属]

CG_rtc.c

[指定形式]

```
#include    "CG_macrodriver.h"
#include    "CG_rtc.h"
MD_STATUS   RTC_CounterGet ( struct RTCCounterValue *counterreadval );
```

[引数]

I/O	引数	説明
O	struct RTCCounterValue *counterreadval;	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 RTCCounterValue についての詳細は，RTC_CounterSet 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 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。

[使用例]

以下に，リアルタイム・カウンタのカウント値を読み出す際の例を示します。

【CG_main.c】

```
#include    "CG_rtc.h"
void main ( void ) {
```

```
    struct  RTCCounterValue counterreadval;
    .........
    RTC_CounterEnable ();               /* カウントの開始 */
    .........
    RTC_CounterGet ( &counterreadval ); /* カウント値の読み出し */
    .........
}
```

RTC_ConstPeriodInterruptEnable

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。

[所属]

CG_rtc.c

[指定形式]

```
#include    "CG_rtc.h"
MD_STATUS   RTC_ConstPeriodInterruptEnable ( enum RTCINTPeriod period );
```

[引数]

I/O	引数	説明
I	enum RTCINTPeriod *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

[使用例]

以下に，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する際の例を示します。

【CG_main.c】

```
#include    "CG_rtc.h"
void main ( void ) {
    .........
    RTC_ConstPeriodInterruptDisable ();          /* 定周期割り込み機能の終了 */
    .........
    RTC_ConstPeriodInterruptEnable ( HALFSEC ); /* 定周期割り込み機能の開始 */
```

```
    .........
}
```

RTC_ConstPeriodInterruptDisable

定周期割り込み機能を終了します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_ConstPeriodInterruptDisable ( void );
```

[引数]

なし

[戻り値]

なし

RTC_ConstPeriodInterruptCallback

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 MD_INTRTC のコールバック・ルーチンとして呼び出されます。

[所属]

CG_rtc_user.c

[指定形式]

```
void    RTC_ConstPeriodInterruptCallback ( void );
```

[引数]

なし

[戻り値]

なし

RTC_AlarmEnable

アラーム割り込み機能を開始します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_AlarmEnable ( void );
```

[引数]

なし

[戻り値]

なし

RTC_AlarmDisable

アラーム割り込み機能を終了します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_AlarmDisable ( void );
```

[引数]

なし

[戻り値]

なし

RTC_AlarmSet

アラームの発生条件（曜日，時，分）を設定します。

[所属]

CG_rtc.c

[指定形式]

```
#include    "CG_rtc.h"
void    RTC_AlarmSet ( struct RTCAlarmValue alarmval );
```

[引数]

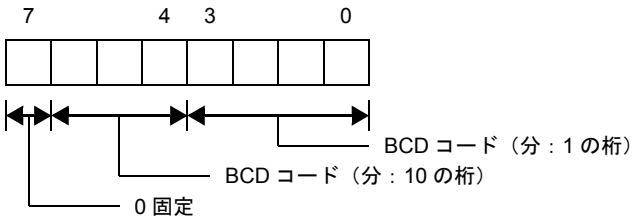
I/O	引数	説明
I	struct RTCAlarmValue *alarmval*;	アラームの発生条件（曜日，時，分）

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

```
struct  RTCAlarmValue {
    UCHAR   Alarmwm;    /* 分 */
    UCHAR   Alarmwh;    /* 時 */
    UCHAR   Alarmww;    /* 曜日 */
};
```

- 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。



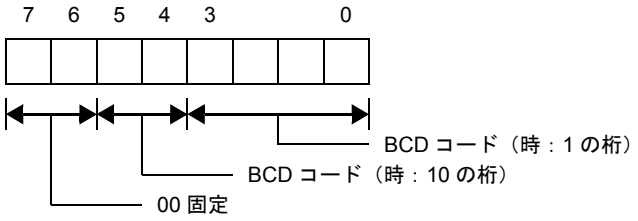
- Alarmwh（時）

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・カウンタが 12 時間制の場合，以下の意味となります。

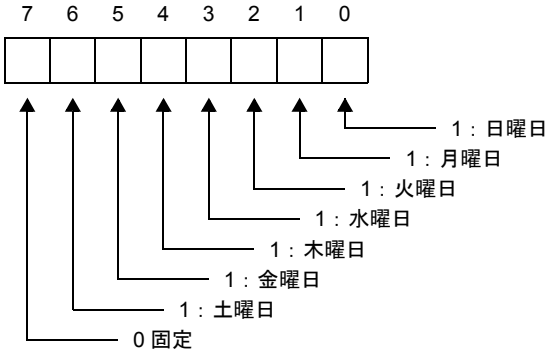
0： 午前

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

[使用例 1]

以下に，アラームの発生条件として，“月曜日／火曜日／水曜日の 17 時 30 分”を設定する際の例を示します。

【CG_main.c】

```
#include    "CG_rtc.h"
void main ( void ) {
    struct  RTCAlarmValue   alarmval;
    ……
    RTC_AlarmEnable ();             /* アラーム割り込み機能の開始 */
    RTC_CounterEnable ();           /* カウントの開始 */
    ……
    RTC_SetHourSystem ( HOUR24 );   /* 時間制の設定 */
    alarmval.Alarmww = 0xe;
    alarmval.Alarmwh = 0x17;
    alarmval.Alarmwm = 0x30;
    RTC_AlarmSet ( alarmval );      /* 発生条件の設定 */
    ……
```

```
}
```

［使用例 2］

以下に，アラームの発生条件を“土曜日／日曜日（時分はそのまま）”に変更する際の例を示します。

【CG_main.c】

```
#include    "CG_rtc.h"
void main ( void ) {
    struct  RTCAlarmValue   alarmval;
    ……
    RTC_AlarmEnable ();             /* アラーム割り込み機能の開始 */
    ……
    alarmval.Alarmww = 0x41;
    RTC_AlarmSet ( alarmval );      /* 発生条件の変更 */
    ……
}
```

RTC_AlarmGet

アラームの発生条件（曜日，時，分）を読み出します。

［所属］

CG_rtc.c

［指定形式］

```
#include    "CG_rtc.h"
void    RTC_AlarmGet ( struct RTCAlarmValue *alarmval );
```

備考 アラームの発生条件 RTCAlarmValue についての詳細は，RTC_AlarmSet を参照してください。

［引数］

I/O	引数	説明
O	struct RTCAlarmValue *alarmval;	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

［使用例］

以下に，アラームの発生条件を読み出す際の例を示します。

【CG_main.c】

```
#include    "CG_rtc.h"
void main ( void ) {
    struct  RTCAlarmValue   alarmval;
    .........
    RTC_AlarmEnable ();         /* アラーム割り込み機能の開始 */
    .........
    RTC_AlarmGet ( &alarmval ); /* 発生条件の読み出し */
    .........
}
```

RTC_AlarmInterruptCallback

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 MD_INTRTC のコールバック・ルーチンとして呼び出されます。

[所属]

CG_rtc_user.c

[指定形式]

```
void    RTC_AlarmInterruptCallback ( void );
```

[引数]

なし

[戻り値]

なし

RTC_IntervalStart

インターバル割り込み機能を開始します。

備考 割り込み INTRTCI の発生周期を設定したのち，インターバル割り込み機能を開始する場合は，RTC_IntervalInterruptEnable を呼び出します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_IntervalStart ( void );
```

[引数]

なし

[戻り値]

なし

RTC_IntervalStop

インターバル割り込み機能を終了します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_IntervalStop ( void );
```

[引数]

なし

[戻り値]

なし

RTC_IntervalInterruptEnable

割り込み INTRTCI の発生周期を設定したのち，インターバル割り込み機能を開始します。

備考 割り込み INTRTCI の発生周期を設定することなく，インターバル割り込み機能を開始する場合は，RTC_IntervalStart を呼び出します。

[所属]

CG_rtc.c

[指定形式]

- 【Ix3（IB3 を除く）】【Kx3-A】【Kx3-L】【Lx3】

```
#include    "CG_rtc.h"
MD_STATUS   RTC_IntervalInterruptEnable ( enum RTCINTInterval interval );
```

- 【Kx3】

```
#include    "CG_rtc.h"
void    RTC_IntervalInterruptEnable ( enum RTCINTInterval interval );
```

[引数]

I/O	引数	説明
I	enum RTCINTInterval *interval*;	割り込み INTRTCI の発生周期 INTERVAL0： 2^6/fXT INTERVAL1： 2^7/fXT INTERVAL2： 2^8/fXT INTERVAL3： 2^9/fXT INTERVAL4： 2^10/fXT INTERVAL5： 2^11/fXT INTERVAL6： 2^12/fXT

備考 fXT は，サブシステム・クロックの周波数を意味します。

[戻り値]

- 【Ix3（IB3 を除く）】【Kx3-A】【Kx3-L】【Lx3】

マクロ	説明
MD_OK	正常終了

マクロ	説明
MD_ARGERROR	引数の指定が不正

- 【Kx3】

  なし

［使用例］

以下に，インターバル間隔を変更したのち，インターバル割り込み機能を再開始する際の例を示します。

【CG_main.c】

```
#include    "CG_rtc.h"
void main ( void ) {
    .........
    RTC_IntervalStart ();                       /* インターバル割り込み機能の開始 */
    .........
    RTC_IntervalStop ();                        /* インターバル割り込み機能の終了 */
    .........
    RTC_IntervalInterruptEnable ( INTERVAL6 );  /* インターバル割り込み機能の開始 */
    .........
}
```

RTC_IntervalInterruptDisable

インターバル割り込み機能を終了します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_IntervalInterruptDisable ( void );
```

[引数]

なし

[戻り値]

なし

RTC_RTC1HZ_OutputEnable

RTC1HZ 端子に対するリアルタイム・カウンタ補正クロック（1 Hz）の出力を許可します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_RTC1HZ_OutputEnable ( void );
```

[引数]

なし

[戻り値]

なし

RTC_RTC1HZ_OutputDisable

RTC1HZ 端子に対するリアルタイム・カウンタ補正クロック（1 Hz）の出力を禁止します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_RTC1HZ_OutputDisable ( void );
```

[引数]

なし

[戻り値]

なし

RTC_RTCCL_OutputEnable

RTCCL 端子に対するリアルタイム・カウンタ・クロック（32 kHz 原発）の出力を許可します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_RTCCL_OutputEnable ( void );
```

[引数]

なし

[戻り値]

なし

RTC_RTCCL_OutputDisable

RTCCL 端子に対するリアルタイム・カウンタ・クロック（32 kHz 原発）の出力を禁止します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_RTCCL_OutputDisable ( void );
```

[引数]

なし

[戻り値]

なし

RTC_RTCDIV_OutputEnable

RTCDIV 端子に対するリアルタイム・カウンタ・クロック（32 kHz 分周）の出力を許可します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_RTCDIV_OutputEnable ( void );
```

[引数]

なし

[戻り値]

なし

RTC_RTCDIV_OutputDisable

RTCDIV 端子に対するリアルタイム・カウンタ・クロック（32 kHz 分周）の出力を禁止します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_RTCDIV_OutputDisable ( void );
```

[引数]

なし

[戻り値]

なし

RTC_ChangeCorrectionValue

時計誤差を補正するタイミング，および補正値を変更します。

[所属]

CG_rtc.c

[指定形式]

```
#include    "CG_macrodriver.h"
#include    "CG_rtc.h"
MD_STATUS   RTC_ChangeCorrectionValue ( enum RTCCorectionTiming timing, UCHAR corectVal );
```

[引数]

I/O	引数	説明
I	enum RTCCorectionTiming *timing*;	時計誤差の補正タイミング EVERY20S： 秒桁が 00，20，40 の時 EVERY60S： 秒桁が 00 の時
I	UCHAR *corectVal*;	時計誤差の補正値

備考 本 API 関数では，補正値 *corectVal* に 0x0，0x1，0x40，または 0x41 が指定された際，時計誤差の補正処理を行いません。

[戻り値]

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

C. 3. 13 クロック出力

以下に，コード生成がクロック出力用として出力する API 関数の一覧を示します。

表 C—14 クロック出力用 API 関数

API 関数名	機能概要
PCL_Init	クロック出力制御回路の機能を制御するうえで必要となる初期化処理を行います。
PCL_UserInit	クロック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。
PCL_Start	クロック出力を開始します。
PCL_Stop	クロック出力を停止します。
PCL_ChangeFreq	PCL 端子への出力クロックを変更します。

PCL_Init

クロック出力制御回路の機能を制御するうえで必要となる初期化処理を行います。

[所属]

CG_pcl.c

[指定形式]

```
void    PCL_Init ( void );
```

[引数]

なし

[戻り値]

なし

PCL_UserInit

クロック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，PCL_Init のコールバック・ルーチンとして呼び出されます。

[所属]

CG_pcl_user.c

[指定形式]

```
void    PCL_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

PCL_Start

クロック出力を開始します。

[所属]

CG_pcl.c

[指定形式]

```
void    PCL_Start ( void );
```

[引数]

なし

[戻り値]

なし

PCL_Stop

クロック出力を停止します。

[所属]

CG_pcl.c

[指定形式]

```
void    PCL_Stop ( void );
```

[引数]

なし

[戻り値]

なし

PCL_ChangeFreq

PCL 端子への出力クロックを変更します。

備考 引数 *clock* に指定された値は，クロック出力選択レジスタ（CKS）に設定されます。

[所属]

CG_pcl.c

[指定形式]

```
#include    "CG_pclbuz.h"
void    PCL_ChangeFreq ( enum PCLclock clock );
```

[引数]

I/O	引数	説明
I	enum PCLclock *clock*;	出力クロックの種類 MAINCLOCK：fMAIN MAIN2：fMAIN/2 MAIN4：fMAIN/4 MAIN8：fMAIN/8 MAIN16：fMAIN/16 MAIN2048：fMAIN/2048 MAIN4096：fMAIN/4096 MAIN8192：fMAIN/8192 PLLCLOCK：fPLL PLL2：fPLL/2 PLL4：fPLL/4 PLL8：fPLL/8 PLL16：fPLL/16 PLL2048：fPLL/2048 PLL4096：fPLL/4096 PLL8192：fPLL/8192 ILCLOCK：fIL SUBCLOCK：fSUB

備考 fMAIN はメイン・システム・クロックの周波数を，fPLL は PLL クロックの周波数を，fIL は低速内蔵発振クロックの周波数を，fSUB はサブシステム・クロックの周波数を意味します。

[戻り値]

なし

C. 3. 14 クロック出力／ブザー出力

以下に，コード生成がクロック出力／ブザー出力用として出力する API 関数の一覧を示します。

表 C—15 クロック出力／ブザー出力用 API 関数

API 関数名	機能概要
PCLBUZn_Init	クロック出力／ブザー出力制御回路の機能を制御するうえで必要となる初期化処理を行います。
PCLBUZn_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。
PCLBUZn_Start	クロック出力／ブザー出力を開始します。
PCLBUZn_Stop	クロック出力／ブザー出力を停止します。
PCLBUZn_ChangeFreq	PCLBUZ*n* 端子への出力クロックを変更します。

PCLBUZ*n*_Init

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

[所属]

CG_pclbuz.c

[指定形式]

```
void    PCLBUZn_Init ( void );
```

備考 *n* は，出力端子を意味します。

[引数]

なし

[戻り値]

なし

PCLBUZ*n*_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，PCLBUZn_Init のコールバック・ルーチンとして呼び出されます。

[所属]

CG_pclbuz_user.c

[指定形式]

```
void    PCLBUZn_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

PCLBUZ*n*_Start

クロック出力／ブザー出力を開始します。

[所属]

CG_pclbuz.c

[指定形式]

```
void    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

PCLBUZ*n*_Stop

クロック出力／ブザー出力を停止します。

[所属]

CG_pclbuz.c

[指定形式]

```
void    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

PCLBUZ*n*_ChangeFreq

PCLBUZ*n* 端子への出力クロックを変更します。

備考 引数 *clock* に指定された値は，クロック出力選択レジスタ *n*（CKS*n*）に設定されます。

[所属]

CG_pclbuz.c

[指定形式]

```
#include    "CG_pclbuz.h"
MD_STATUS   PCLBUZn_ChangeFreq ( enum PCLBUZclock clock );
```

備考 *n* は，出力端子を意味します。

[引数]

I/O	引数	説明
I	enum PCLBUZclock *clock*;	出力クロックの種類 MAINCLOCK：fMAIN MAIN2：fMAIN/2 MAIN4：fMAIN/4 MAIN8：fMAIN/8 MAIN16：fMAIN/16 MAIN2048：fMAIN/2048 MAIN4096：fMAIN/4096 MAIN8192：fMAIN/8192 SUBCLOCK：fSUB SUB2：fSUB/2 SUB4：fSUB/4 SUB8：fSUB/8 SUB16：fSUB/16 SUB32：fSUB/32 SUB64：fSUB/64 SUB128：fSUB/128

備考 fMAIN はメイン・システム・クロックの周波数を，fSUB はサブシステム・クロックの周波数を意味します。

[戻り値]

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

[使用例]

以下に，外部接続のスイッチ SW を押下した際に割り込み INTP0 が発生するシステムにおいて，スイッチを押下するたびに PCLBUZ0 端子への出力クロックを変更する例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
#include    "CG_pclbuz.h"
BOOL    gFlag;                                  /* SW 押下フラグ */
#define PCLBUZ_FREQUENCY    8                   /* 出力クロックの種類総数 */
const   enum    PCLBUZclock gClock[PCLBUZ_FREQUENCY] = {    /* 出力クロックの種類 */
    PCLBUZ_OUTCLK_fSUB0, PCLBUZ_OUTCLK_fSUB1, PCLBUZ_OUTCLK_fSUB2,
    PCLBUZ_OUTCLK_fSUB3, PCLBUZ_OUTCLK_fSUB4, PCLBUZ_OUTCLK_fSUB5,
    PCLBUZ_OUTCLK_fSUB6, PCLBUZ_OUTCLK_fSUB7
};
void main ( void ) {
    int     index = 0;
    gFlag = 0;                                  /* SW 押下フラグの初期化 */
    ……
    PCLBUZ0_Start ();                           /* クロック出力／ブザー出力の開始 */
    while ( 1 ) {
        if ( gFlag ) {                          /* 割り込み INTP0 の発生待ち */
            PCLBUZ_ChangeFreq ( gClock[index++] );  /* 出力クロックの変更 */
            if ( index >= PCLBUZ_FREQUENCY ) {
                index = 0;
            }
            gFlag = 0;                          /* SW 押下フラグのクリア */
        }
    }
}
```

【CG_int_user.c】

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                      /* SW 押下フラグ */
__interrupt void MD_INTP0 ( void ) {        /* 割り込み INTP0 発生時の割り込み処理 */
    gFlag = 1;                              /* SW 押下フラグの設定 */
}
```

C. 3. 15 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—16 LCD コントローラ／ドライバ用 API 関数

API 関数名	機能概要
LCD_Init	LCD コントローラ／ドライバの機能を制御するうえで必要となる初期化処理を行います。
LCD_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。
LCD_DisplayOn	LCD コントローラ／ドライバを表示オン状態にします。
LCD_DisplayOff	LCD コントローラ／ドライバを表示オフ状態にします。
LCD_VoltageOn	LCD コントローラ／ドライバの昇圧回路，および容量分割回路を動作許可としたのち，非選択信号をセグメント端子から出力します。
LCD_VoltageOff	LCD コントローラ／ドライバの昇圧回路，および容量分割回路を動作停止としたのち，グランド・レベルの信号をセグメント端子／コモン端子に出力します。

LCD_Init

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

[所属]

CG_lcd.c

[指定形式]

```
void    LCD_Init ( void );
```

[引数]

なし

[戻り値]

なし

LCD_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，LCD_Init のコールバック・ルーチンとして呼び出されます。

[所属]

CG_lcd_user.c

[指定形式]

```
void    LCD_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

LCD_DisplayOn

LCD コントローラ／ドライバを表示オン状態にします。

[所属]

CG_lcd.c

[指定形式]

```
void    LCD_DisplayOn ( void );
```

[引数]

なし

[戻り値]

なし

LCD_DisplayOff

LCD コントローラ／ドライバを表示オフ状態にします。

[所属]

CG_lcd.c

[指定形式]

```
void    LCD_DisplayOff ( void );
```

[引数]

なし

[戻り値]

なし

LCD_VoltageOn

LCD コントローラ／ドライバの昇圧回路，および容量分割回路を動作停止としたのち，非選択信号をセグメント端子から出力します。

[所属]

CG_lcd.c

[指定形式]

```
void    LCD_VoltageOn ( void );
```

[引数]

なし

[戻り値]

なし

LCD_VoltageOff

LCD コントローラ／ドライバの昇圧回路，および容量分割回路を動作停止としたのち，グランド・レベルの信号をセグメント端子／コモン端子から出力します。

[所属]

CG_lcd.c

[指定形式]

```
void    LCD_VoltageOff ( void );
```

[引数]

なし

[戻り値]

なし

C. 3. 16 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—17 DMA コントローラ用 API 関数

API 関数名	機能概要
DMAn_Init	DMA コントローラの機能を制御するうえで必要となる初期化処理を行います。
DMAn_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。
DMAn_Enable	チャネル *n* を動作許可状態に設定します。
DMAn_Disable	チャネル *n* を動作停止状態に設定します。
DMAn_Hold	DMA 起動要求を保留します。
DMAn_Restart	DMA 起動要求の保留を解除します。
DMAn_CheckStatus	転送状態（転送終了，転送中）を読み出します。
DMAn_SetData	転送先／転送元の RAM アドレス，およびデータの転送回数を設定します。
DMAn_SoftwareTriggerOn	DMA 動作許可状態の際，DMA 転送を開始します。

DMA*n*_Init

DMA コントローラの機能を制御するうえで必要となる初期化処理を行います。

[所属]

CG_dma.c

[指定形式]

```
void    DMAn_Init ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

DMA*n*_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，DMAn_Init のコールバック・ルーチンとして呼び出されます。

[所属]

CG_dma_user.c

[指定形式]

```
void    DMAn_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

DMA*n*_Enable

チャネル *n* を動作許可状態に設定します。

[所属]

CG_dma.c

[指定形式]

```
void    DMAn_Enable ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

DMA*n*_Disable

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，DMAn_CheckStatus による“転送終了”の確認後に呼び出す必要があります。

[所属]

CG_dma.c

[指定形式]

```
void    DMAn_Disable ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

[使用例]

以下に，チャネル 0 の動作モードを“動作停止状態”へと移行させる際の例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
void main ( void ) {
    .........
    while ( MD_COMPLETED == DMA0_CheckStatus () );  /* 転送状態の確認 */
    DMA0_Disable ();                                /* 動作停止状態への移行 */
    .........
}
```

DMA*n*_Hold

DMA 起動要求を保留します。

備考 DMA 起動要求の保留を解除する際は，DMAn_Restart を呼び出します。

[所属]

CG_dma.c

[指定形式]

```
void    DMAn_Hold ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

DMA*n*_Restart

DMA 起動要求の保留を解除します。

[所属]

CG_dma.c

[指定形式]

```
void    DMAn_Restart ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

DMA*n*_CheckStatus

転送状態（転送終了，転送中）を読み出します。

［所属］

CG_dma.c

［指定形式］

```
#include    "CG_macrodriver.h"
MD_STATUS   DMAn_CheckStatus ( void );
```

備考 *n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

マクロ	説明
MD_UNDEREXEC	転送中
MD_COMPLETED	転送終了

DMA*n*_SetData

転送先／転送元の RAM アドレス，およびデータの転送回数を設定します。

備考 本 API 関数を転送中に呼び出した場合，転送は強制終了します。

［所属］

CG_dma.c

［指定形式］

- 【Fx3】【Ix3】【Kx3-A】【Kx3-L】【Lx3】

```
#include    "CG_macrodriver.h"
MD_STATUS   DMAn_SetData ( UCHAR sfraddr, USHORT ramaddr, USHORT count );
```

- 【Kx3】

```
#include    "CG_macrodriver.h"
MD_STATUS   DMAn_SetData ( USHORT ramaddr, USHORT count );
```

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	UCHAR *sfraddr*;	転送先／転送元の SFR アドレス
I	USHORT *ramaddr*;	転送先／転送元の RAM アドレス
I	USHORT *count*;	データの転送回数（1 ～ 1024）

［戻り値］

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

DMA*n*_SoftwareTriggerOn

DMA 動作許可状態の際，DMA 転送を開始します。

[所属]

CG_dma.c

[指定形式]

```
void    DMAn_SoftwareTriggerOn ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

[使用例]

以下に，ソフトウエア・トリガを DMA 転送の起動要因とした場合の例を示します。

【CG_main.c】

```
void main ( void ) {
    .........
    DMA0_Enable ();              /* 動作許可状態への移行 */
    DMA0_SoftwareTriggerOn ();   /* DMA 転送の開始 */
    .........
}
```

C. 3. 17 低電圧検出回路

以下に，コード生成が低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—18 低電圧検出回路用 API 関数

API 関数名	機能概要
LVI_Init	低電圧検出回路の機能を制御するうえで必要となる初期化処理を行います。
LVI_UserInit	低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。
LVI_InterruptModeStart	低電圧検出動作を開始します（割り込み発生モード時）。
LVI_ResetModeStart	低電圧検出動作を開始します（内部リセット・モード時）。
LVI_Stop	低電圧検出動作を停止します。
LVI_SetLVILevel	低電圧検出レベルを設定します。

LVI_Init

低電圧検出回路の機能を制御するうえで必要となる初期化処理を行います。

[所属]

CG_lvi.c

[指定形式]

```
void    LVI_Init ( void );
```

[引数]

なし

[戻り値]

なし

LVI_UserInit

低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，LVI_Init のコールバック・ルーチンとして呼び出されます。

[所属]

CG_lvi_user.c

[指定形式]

```
void    LVI_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

LVI_InterruptModeStart

低電圧検出動作を開始します（割り込み発生モード時）。

[所属]

CG_lvi.c

[指定形式]

```
void    LVI_InterruptModeStart ( void );
```

[引数]

なし

[戻り値]

なし

[使用例]

以下に，低電圧を検出した際の動作モードが割り込み発生モード（割り込み INTLVI を発生させる）における例を示します。

【CG_main.c】

```
void main ( void ) {
    ……
    LVI_InterruptModeStart ( );          /* 低電圧検出動作の開始 */
    ……
}
```

【CG_lvi_user.c】

```
__interrupt void MD_INTLVI ( void ) {    /* 割り込み INTLVI 発生時の割り込み処理 */
    if ( LVIF == 1 ) {                   /* 発生要因の判別：LVIF フラグのチェック */
        ……    /* “電源電圧（VDD）< 検出電圧（VLVI）”を検出した際の処理 */
    } else {
        ……    /* “電源電圧（VDD）≧ 検出電圧（VLVI）”を検出した際の処理 */
    }
}
```

LVI_ResetModeStart

低電圧検出動作を開始します（内部リセット・モード時）。

[所属]

CG_lvi.c

[指定形式]

```
MD_STATUS   LVI_ResetModeStart ( void );
```

[引数]

なし

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR	異常終了 - 低電圧検出回路の機能を使用しない設定が行われている - 低電圧検出対象が外部電圧（V_{DD}）の際，電源電圧（V_{DD}）≦ 検出電圧（V_{LVI}） - 低電圧検出対象が外部入力電圧（EXLVI）の際，外部入力電圧（EXLVI）≦ 検出電圧（V_{EXLVI}）

LVI_Stop

低電圧検出動作を停止します。

[所属]

CG_lvi.c

[指定形式]

```
void    LVI_Stop ( void );
```

[引数]

なし

[戻り値]

なし

LVI_SetLVILevel

低電圧検出レベルを設定します。

備考 1. 低電圧検出レベルを変更する際には，本 API 関数の呼び出し以前に LVI_Stop を呼び出す必要があります。

2. 引数 *level* に指定された値は，低電圧検出レベル選択レジスタ（LVIS）に設定されます。

[所属]

CG_lvi.c

[指定形式]

```
#include    "CG_macrodriver.h"
#include    "CG_lvi.h"
MD_STATUS   LVI_SetLVILevel ( enum LVILevel level );
```

[引数]

I/O	引数	説明
I	enum LVILevel *level*;	低電圧検出対象の電圧レベル LVILEVEL0： 4.22 V ± 0.1 V LVILEVEL1： 4.07 V ± 0.1 V LVILEVEL2： 3.92 V ± 0.1 V LVILEVEL3： 3.76 V ± 0.1 V LVILEVEL4： 3.61 V ± 0.1 V LVILEVEL5： 3.45 V ± 0.1 V LVILEVEL6： 3.30 V ± 0.1 V LVILEVEL7： 3.15 V ± 0.1 V LVILEVEL8： 2.99 V ± 0.1 V LVILEVEL9： 2.84 V ± 0.1 V LVILEVEL10： 2.68 V ± 0.1 V LVILEVEL11： 2.53 V ± 0.1 V LVILEVEL12： 2.38 V ± 0.1 V LVILEVEL13： 2.22 V ± 0.1 V LVILEVEL14： 2.07 V ± 0.1 V LVILEVEL15： 1.91 V ± 0.1 V

備考 対象デバイスが 78K0R/Fx3，または 78K0R/Ix3 の場合，*level* に指定可能な値は LVILEVEL0 ～ LVILEVEL9 に限られます。

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR	異常終了
MD_ARGERROR	引数の指定が不正

付録D 索 引

【さ行】

【わ行】

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2012.09.01	―	初版発行

CubeSuite+ V1.03.00　ユーザーズマニュアル
78K0R 設計編

発行年月日　2012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/

Colophon 1.1

CubeSuite+ V1.03.00

ルネサス エレクトロニクス株式会社

R20UT2137JJ0100