User's Manual

CubeSuite+ V1.03.00

Integrated Development Environment

User's Manual: 78K0 Design

Target Device

78K0 Microcontrollers

All information contained in these matenals, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website (http://www.renesas.com).

Renesas Electronics
www.renesas.com

Rev.1.00 Sep 2012

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.

   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(2012.4)

How to Use This Manual

This manual describes the role of the CubeSuite+ integrated development environment for developing applications and systems for 78K0 microcontrollers, and provides an outline of its features.

CubeSuite+ is an integrated development environment (IDE) for 78K0 microcontrollers, integrating the necessary tools for the development phase of software (e.g. design, implementation, and debugging) into a single platform.

By providing an integrated environment, it is possible to perform all development using just this product, without the need to use many different tools separately.

Readers

This manual is intended for users who wish to understand the functions of the CubeSuite+ and design software and hardware application systems.

Purpose

This manual is intended to give users an understanding of the functions of the CubeSuite+ to use for reference in developing the hardware or software of systems using these devices.

Organization

This manual can be broadly divided into the following units.

How to Read This Manual

It is assumed that the readers of this manual have general knowledge of electricity, logic circuits, and microcontrollers.

Conventions

Data significance:	Higher digits on the left and lower digits on the right
Active low representation:	$\overline{\text{XXX}}$ (overscore over pin or signal name)
Note:	Footnote for item marked with Note in the text
Caution:	Information requiring particular attention
Remark:	Supplementary information
Numeric representation:	Decimal ... XXXX
	Hexadecimal ... 0xXXXX

Related Documents

The related documents indicated in this publication may include preliminary versions. However, preliminary versions are not marked as such.

Document Name		Document No.
CubeSuite+ Integrated Development Environment User's Manual	Start	R20UT2133E
	V850 Design	R20UT2134E
	R8C Design	R20UT2135E
	RL78 Design	R20UT2136E
	78K0R Design	R20UT2137E
	78K0 Design	This manual
	RX Coding	R20UT0767E
	V850 Coding	R20UT0553E
	Coding for CX Compiler	R20UT2139E
	R8C Coding	R20UT0576E
	RL78, 78K0R Coding	R20UT2140E
	78K0 Coding	R20UT2141E
	RX Build	R20UT0768E
	V850 Build	R20UT0557E
	Build for CX Compiler	R20UT2142E
	R8C Build	R20UT0575E
	RL78, 78K0R Build	R20UT2143E
	78K0 Build	R20UT0783E
	RX Debug	R20UT2175E
	V850 Debug	R20UT2144E
	R8C Debug	R20UT0770E
	RL78 Debug	R20UT2145E
	78K0R Debug	R20UT0732E
	78K0 Debug	R20UT0731E
	Analysis	R20UT2146E
	Message	R20UT2147E

Caution The related documents listed above are subject to change without notice. Be sure to use the latest edition of each document when designing.

All trademarks or registered trademarks in this document are the property of their respective owners.

TABLE OF CONTENTS

CHAPTER 1 GENERAL

CubeSuite+ is an integrated development environment used to carry out tasks such as design, coding, build and debug for developing application systems.

This chapter gives an overview of the design tool (Pin Configurator/Code Generator).

1.1 Overview

The design tool, which is one of the components provided by CubeSuite+, enables you to output the pin assignment of the microcontroller (device pin list and device top view), and the source code (device driver programs, C source files and header files) necessary to control the peripheral functions provided by the microcontroller (clock generator, port functions, etc.) by configuring various information using the GUI.

1.2 Features

The design tool (Pin Configurator/Code Generator) has the following features.

- Code generating function
  The Code Generator can output not only device driver programs in accordance with the information configured using the GUI, but also a build environment such as sample programs containing main functions and link directive files.

- Reporting function
  You can output configured information using Pin Configurator/Code Generator as files in various formats for use as design documents.

- Renaming function
  The user can change default names assigned to the files output by Code Generator and the API functions contained in the source code.

CHAPTER 2 FUNCTIONS (Pin Configurator)

This chapter describes the key functions provided by the design tool (Pin Configurator) along with operation procedures.

2.1 Overview

The Pin Configurator is used to output report files such as a device pin list and a device top view by entering pin assignment information of the microcontroller.

The following sections describe the operation procedures for Pin Configurator.

(1) Start CubeSuite+

Launch CubeSuite+ from the [Start] menu of Windows.

Remark See "CubeSuite+ Integrated Development Environment User's Manual: Start" for details on "Start CubeSuite+".

(2) Create/Open project

Create a new project (that defines a kind of project, microcontroller to be used, build tools to be used, etc.) or load an existing project.

Remark See "CubeSuite+ Integrated Development Environment User's Manual: Start" for details on "Create/Open project".

(3) Open Device Pin List Panel

Open the Device Pin List panel, where you enter information on the pins of the microcontroller.

(a) Select item

Allows you to select items displayed in the device pin list.

(b) Change display order

Allows you to change the order in which items are displayed in the device pin list.

(c) Add column

Allows you to add columns to the device pin list.

(d) Delete column

Allows you to delete columns from the device pin list.

(4) Open Device Top View Panel

Open the Device Top View panel, where you can confirm the information entered for the pins.

(a) Select shape of microcontroller

Allows you to select the shape of the microcontroller displayed in the Device Top View panel.

(b) Select color

Allows you to select colors used to distinguish the type of pins (power pins, special pins, used pins, etc.) whose information is displayed in the Device Top View panel.

(c) Select popup information

Allows you to select the type of information that popups when you move the mouse cursor over each pin in the Device Top View panel.

(d) Select additional information

Select the type of information to display in Pin area of the Device Top View panel.

(5) Enter Information

Enter information on the pins of the microcontroller in the Device Pin List panel.

(6) Output Report Files

Output report files (files containing configured information using Pin Configurator: device pin list and device top view) to the specified folder.

(a) Output device pin list

Output a device pin list.

(b) Output device top view

Output a device top view.

(7) Save project

Save a project.

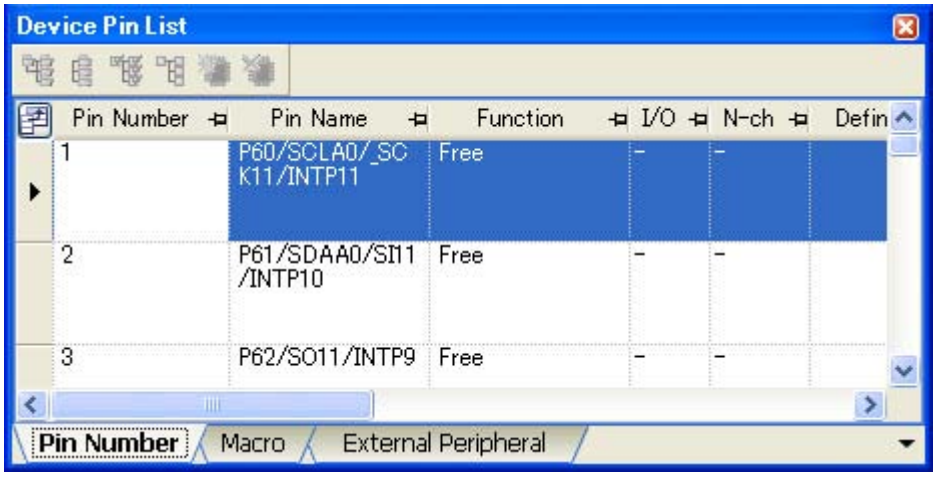
Remark See "CubeSuite+ Integrated Development Environment User's Manual: Start" for details on "Save project".

2.2 Open Device Pin List Panel

Open the Device Pin List panel, where you enter information on the pins of the microcontroller.

To open the Device Pin List panel, double-click [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Pin List] in the Project Tree panel.

Figure 2-1. Open Device Pin List Panel



Remarks 1. If an unsupported microcontroller is defined in the project for Pin Configurator, then "[Pin Configurator (Design Tool)] node" will hide under [*Project name* (Project)] in the Project Tree panel.

2. The Device Pin List panel consists of three tabs. Selecting one of the tabs changes the order in which "information on each pin of the microcontroller" is displayed.

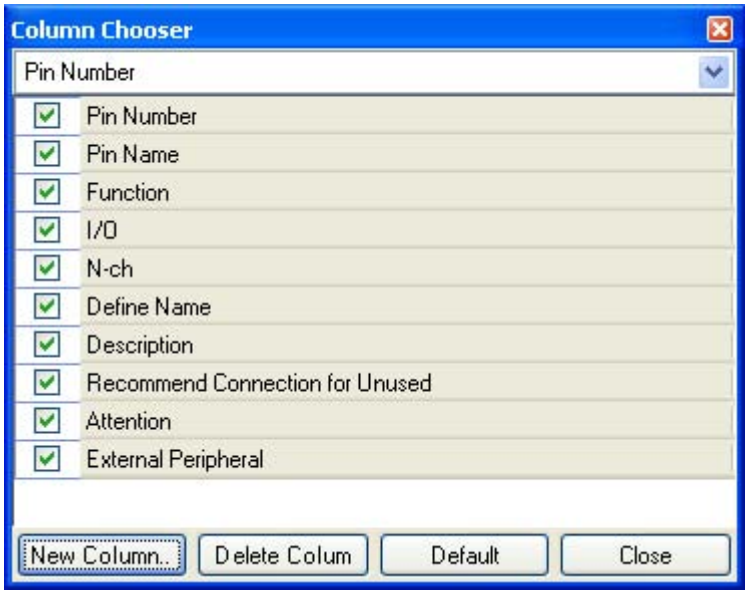
- [Pin Number] tab
  Information on each pin of the microcontroller is displayed in the order of pin number.
- [Macro] tab
  Information on each pin of the microcontroller is displayed in the order it was grouped into peripheral functions.
- [External Peripheral] tab
  Information about the pins connected to external peripherals is displayed in order grouped at the external-peripheral component level.

2.2.1 Select item

The Pin Configurator is used to select items to be displayed in the device pin list using the button in the upper left corner of the device pin list.

To select the item to be displayed, use the Column Chooser dialog box that opens by pressing the button in the upper left corner of the device pin list.

Figure 2-2. Select Item



Remark To select the item to be displayed, check the check box that corresponds to the item.

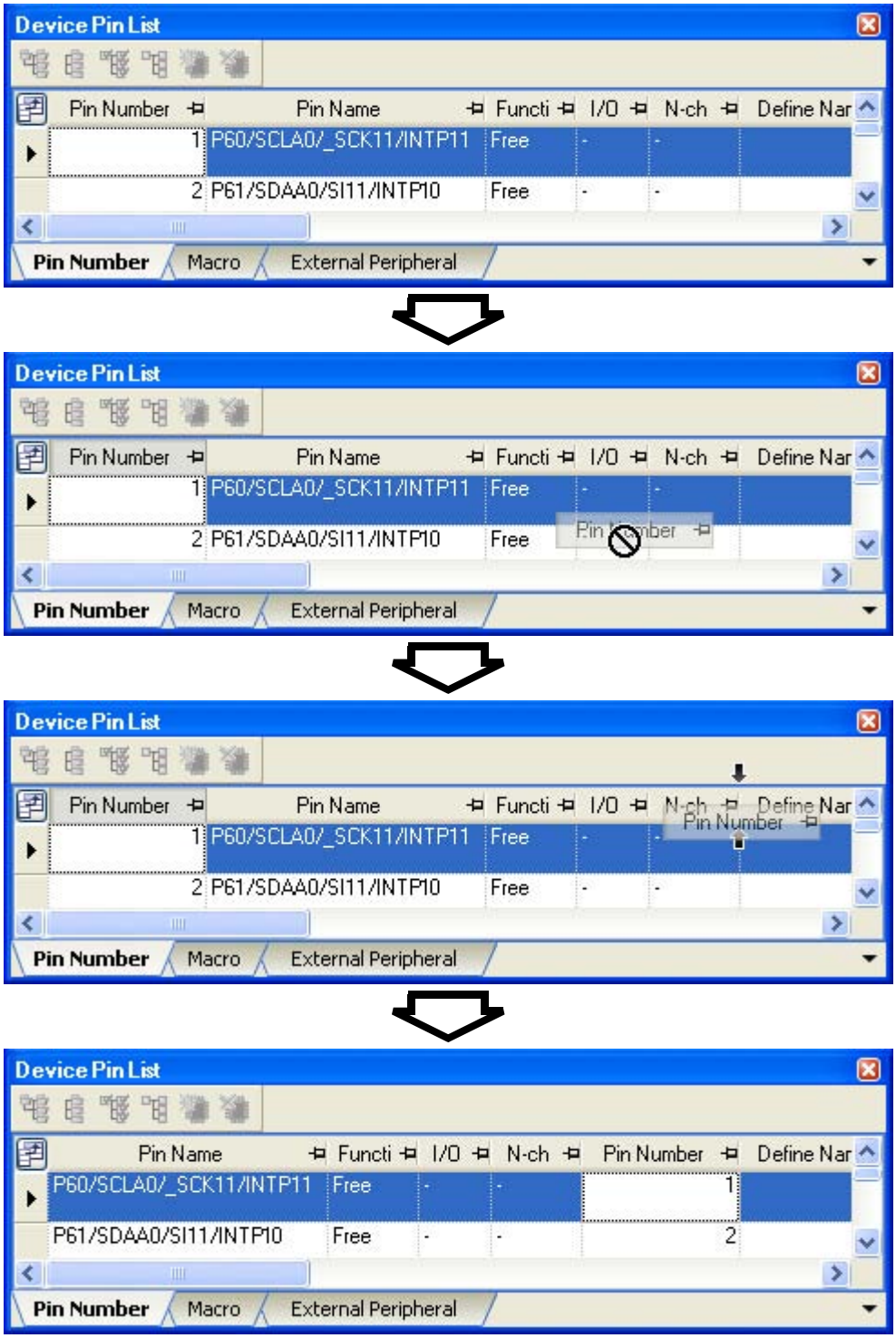
Table 2-1. Select Item

Checked	Displays the selected item in the device pin list.
Not checked	Hides the selected item in the device pin list.

2.2.2 Change display order

In Pin Configurator, you can change the display order of columns in the device pin list (move columns) by dragging and dropping columns.

Figure 2-3. Change Display Order



Remark To change the display order, click the button in the upper left of the device pin list. The Column Chooser dialog box opens. Drag an item displayed in the dialog's select Items to display area, and drop it to the desired destination in the device pin list. This will change the display order.

2.2.3 Add column

The Pin Configurator is used to add the "user's own column" to the device pin list using the [New Column...] button in the Column Chooser dialog box that opens by pressing the button in the upper left corner of the device pin list.

To add a column, use the New Column dialog box that opens by pressing the [New Column...] button in the Column Chooser dialog box.

Figure 2-4. Add Column

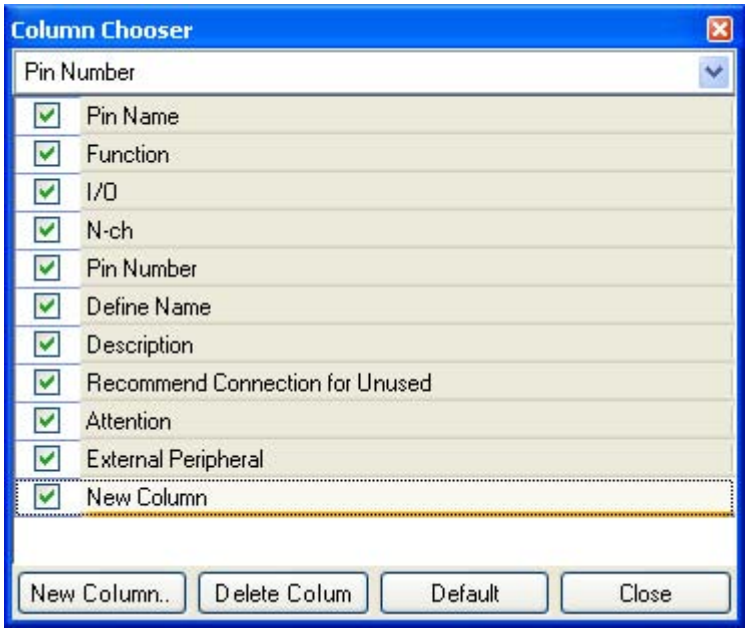
Remark On the device pin list, adding columns to the first level of [Macro] tab, [External Peripheral] tab is restricted.

2.2.4 Delete column

The Pin Configurator is used to delete the "user's own column" from the device pin list using the [Delete Column] button in the Column Chooser dialog box that opens by pressing the button in the upper left corner of the device pin list.

To delete a column, select the column you want to delete in the displayed item selection area of the Column Chooser dialog box, and press the [Delete Column] button.

Figure 2-5. Delete Column



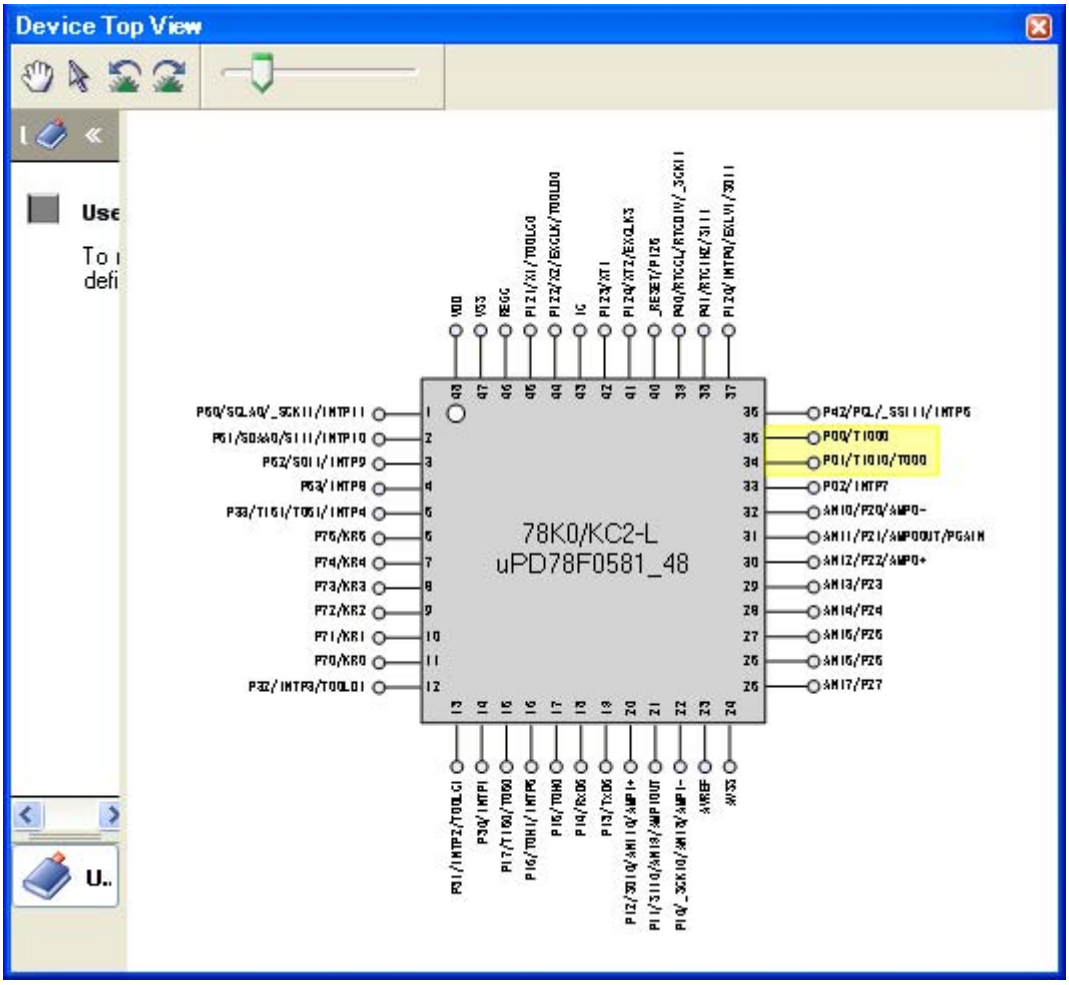
Remark You can only delete the column which you added using the New Column dialog box.

2.3 Open Device Top View Panel

Open the Device Top View panel, where you can confirm the information entered for the pins of the microcontroller.

To open the Device Top View panel, double-click [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Top View] in the Project Tree panel.

Figure 2-6. Open Device Top View Panel



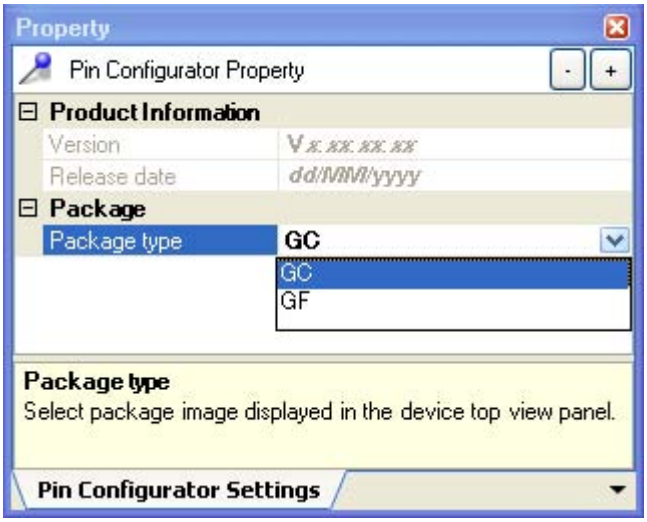
Remark In the Property panel, on the [Pin Configurator Settings] tab, if "BGA" is selected for the Package type, then Device Top View panel cannot be opened.

2.3.1 Select shape of microcontroller

Select the shape of the microcontroller displayed in the Device Top View panel which is opened as described in "2.3 Open Device Top View Panel".

To select the shape of the microcontroller, click [Pin Configurator Settings] tab >> [Package type] in the Property panel and select the desired shape.

Figure 2-7. Select Shape of Microcontroller



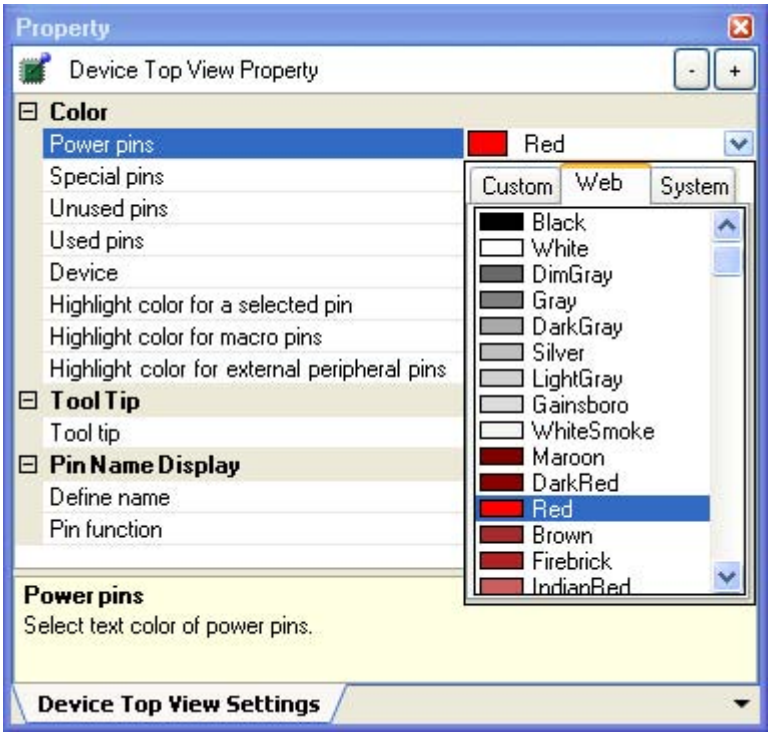
Remark Selection of the shape of the microcontroller is made using the order name (such as GC and GF).

2.3.2 Select color

Select the colors used to distinguish the type of pins (power pins, special pins, unused pins, etc.) whose information is displayed in the Device Top View panel which is opened as described in "2.3 Open Device Top View Panel".

To select the color to be displayed, select the desired color in the color palette that opens by clicking [Device Top View Settings] tab >> [Color] in the Property panel.

Figure 2-8. Select Color



Remark Select the colors to be displayed for the following eight types of items.

Table 2-2. Select Color

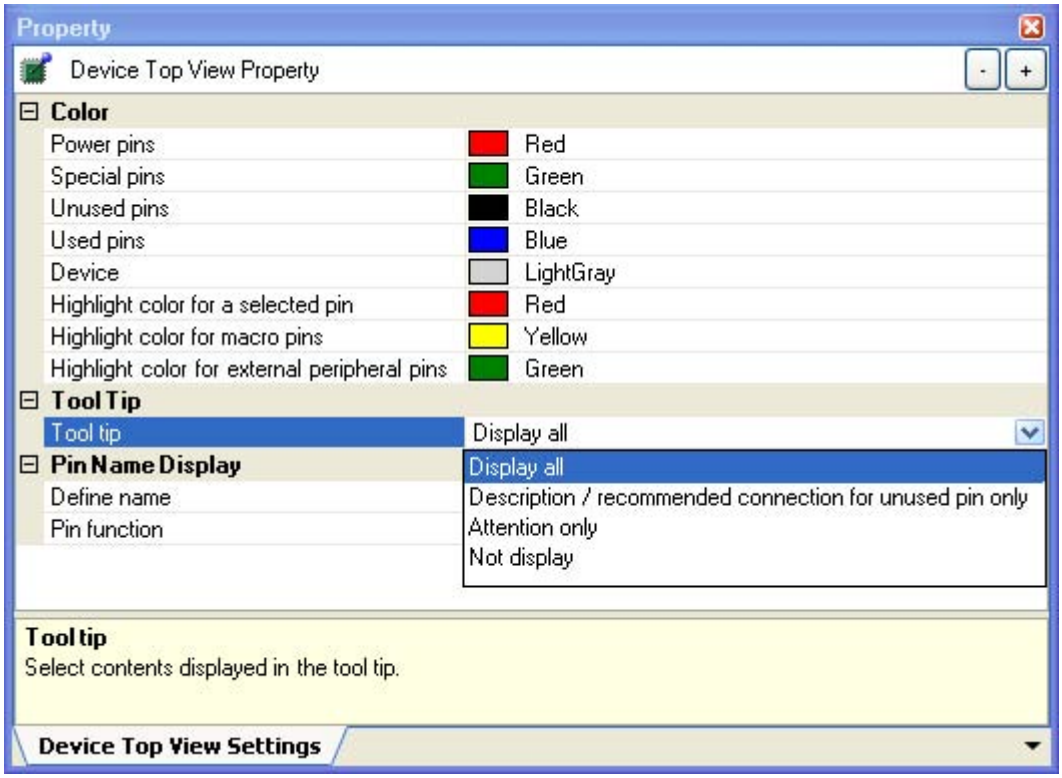
Item	Outline
Power pins	Selects the display color for power pins (pins whose use is limited to power).
Special pins	Selects the display color for special pins (pins with specified uses).
Unused pins	Selects the display color for unused pins (dual-use pins with no use set in the Device Pin List panel).
Used pins	Selects the display color for used pins (dual-use pins with a use set in the Device Pin List panel).
Device	Selects the display color of the microcontroller.
Highlight color for a selected pin	Selects the background color of a pin selected in the Device Pin List panel, on the [Pin Number] tab.
Highlight color for macro pins	Selects the background color of pins selected in the Device Pin List panel, on the [Macro] tab.
Highlight color for external peripheral pins	Selects the background color of pins selected in the Device Pin List panel, on the [External Peripheral] tab.

2.3.3 Select popup information

Select the type of information that popups when you move the mouse cursor over each pin in the Device Top View panel which is opened as described in "2.3 Open Device Top View Panel".

To select the popup information, click [Device Top View Settings] tab >> [Tool tip] in the Property panel and select the desired type of information.

Figure 2-9. Select Popup Information



Remark Popup information is selected from the following four types.

Table 2-3. Select Popup Information

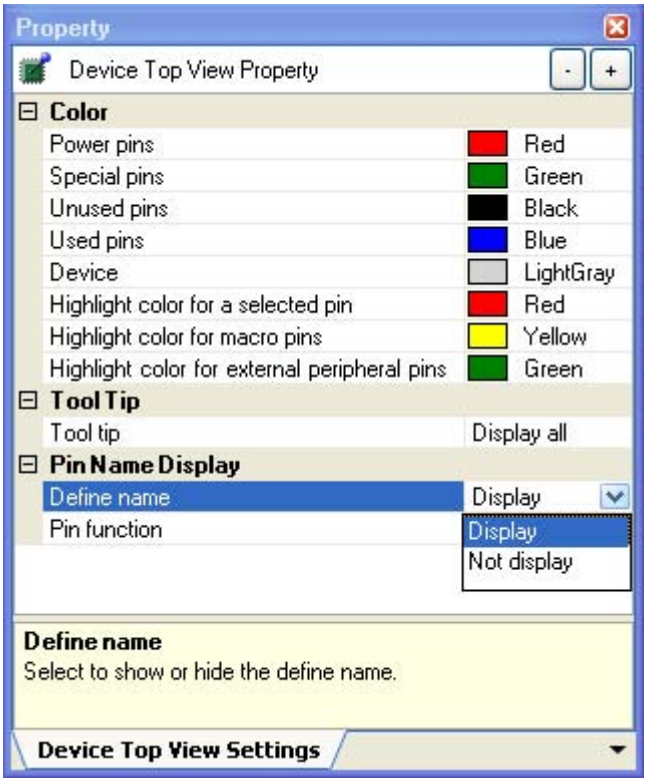
Popup Information	Outline
Display all	Displays the "Description", "Recommend Connection for Unused", and "Attention" strings for the device pin list.
Description / recommended connection for unused pin only	Displays the "Description", and "Recommend Connection for Unused" strings for the device pin list.
Attention only	Displays the "Attention" string for the device pin list.
Not display	Hides tooltips when the mouse cursor hovers over a pin.

2.3.4 Select additional information

Select the type of information to display in Pin area, in the Device Top View panel opened in "2.3 Open Device Top View Panel".

Note that additional information is selected from the Property panel, on the [Device Top View Settings] tab, by selecting the corresponding information under [Pin Name Display].

Figure 2-10. Select Additional Information



Remarks 1. Select one of the following two types for Define name (whether to display the "Define Name" string of the Device Pin List in appended format).

Display	Displays the "Define Name" string of the device pin list in appended format.
Not display	Hides the "Define Name" string of the device pin list.

2. Select one of the following two types for Pin function (whether to display it whether or not a function is selected for "Function" on the Device Pin List).

Display all	Displays functions selected via the device pin list's "Function" feature in parentheses.
Selected function only	Only display functions selected via the device pin list's "Function" feature in the device top view.

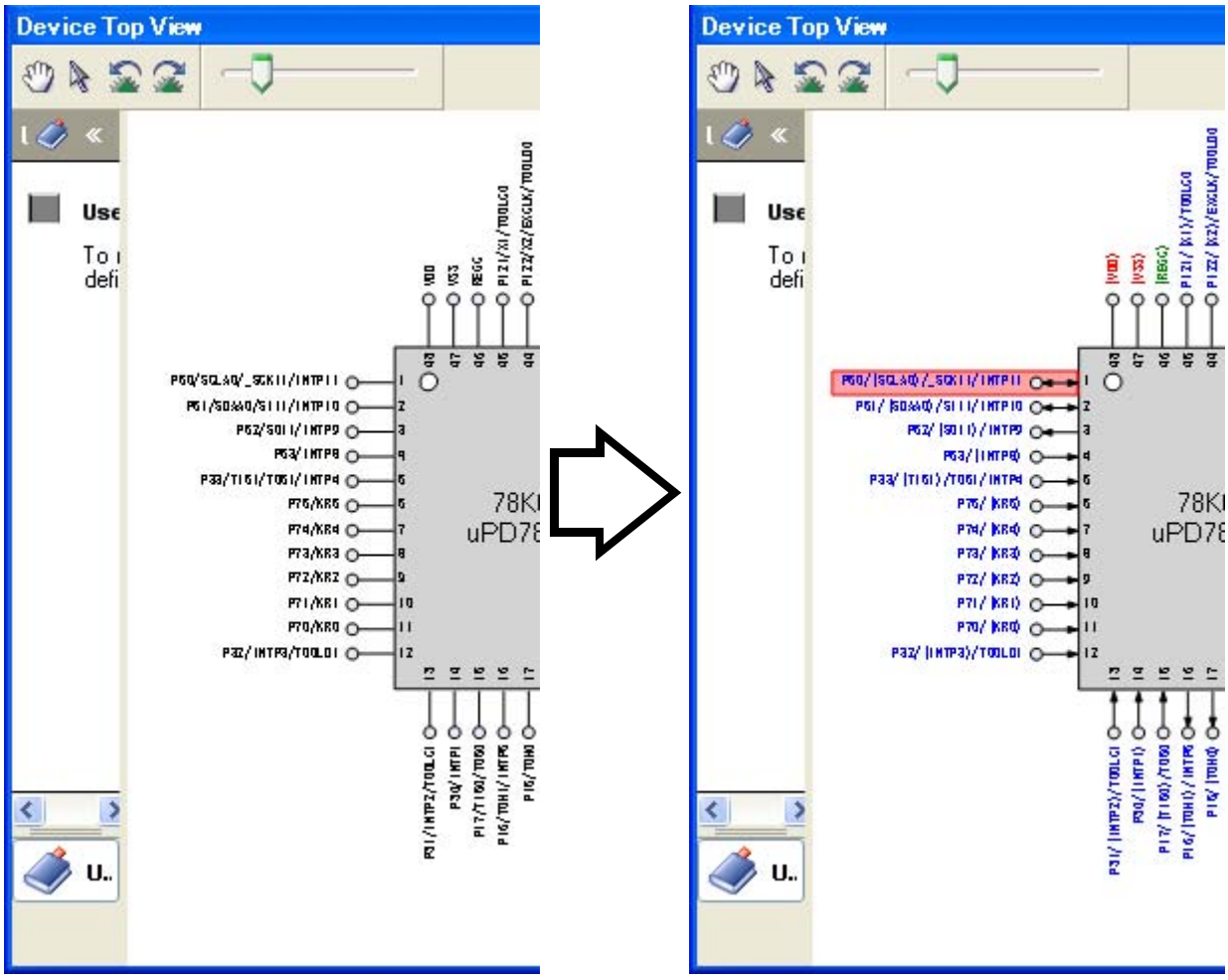
2.4 Enter Information

Enter information on the pins of the microcontroller in the Device Pin List panel which is opened as described in "2.2 Open Device Pin List Panel".

Remarks 1. You cannot add information in the "Pin Number" column, "Pin Name" column, "Description" column, "Recommend Connection for Unused" column and "Attention" column because they contain fixed information.

2. If the "Free" in the "Function" column is changed to a specific pin name, color of the corresponding pin in the Device Top View panel changes from the "color representing the unused pins" to the "color representing the used pins" selected by clicking [Device Top View Settings] tab >> [Color] in the Property panel.

Figure 2-11. Change in Displayed Color



2.5 Output Report Files

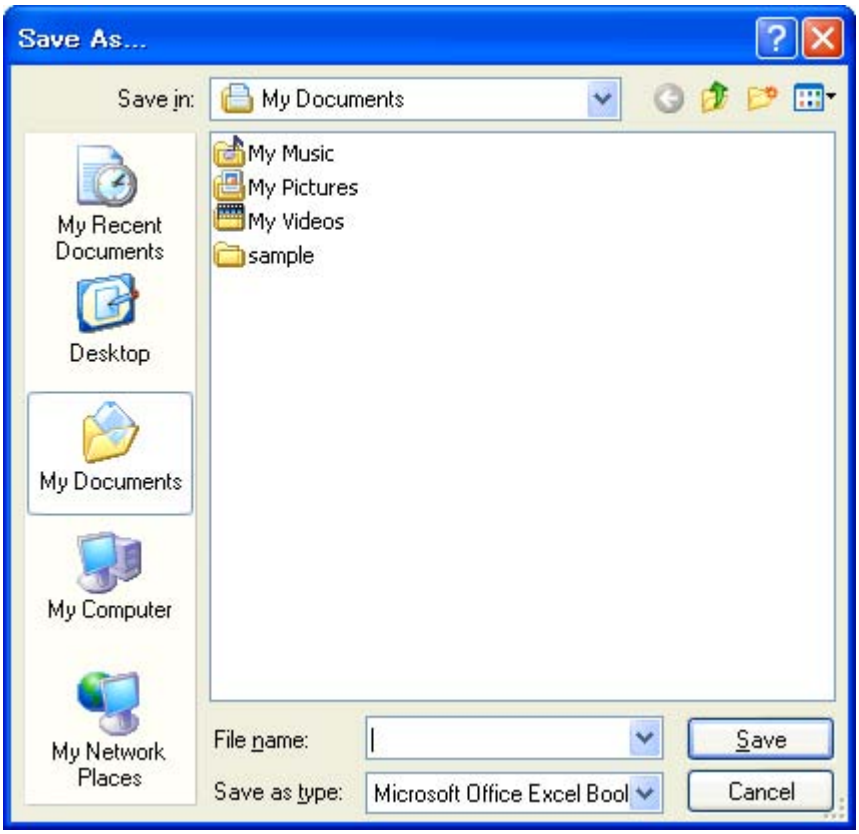
Output report files (files containing information configured using Pin Configurator: device pin list and device top view) to the specified folder.

2.5.1 Output device pin list

Select [File] menu >> [Save Pin List As...] to output a report file (a file containing information configured using Pin Configurator: device pin list).

The destination folder for the device pin list is specified in the Save As dialog box which opens by selecting [File] menu >> [Save Pin List As ...].

Figure 2-12. Output Device Pin List



Remarks 1. If a device pin list has been already output, that list will be overwritten by selecting [File] menu >> [Save Pin List].

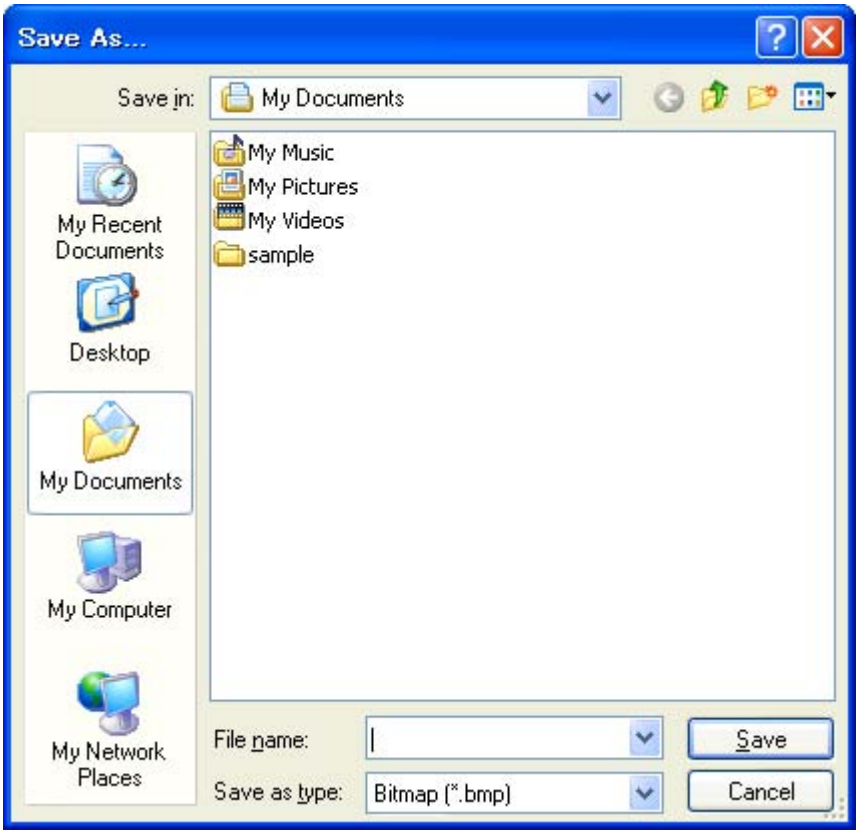
2. The output format for the device pin list is limited to Microsoft Office Excel Book.

2.5.2 Output device top view

Select [File] menu >> [Save Top View As...] to output a report file (a file containing information configured using Pin Configurator: device top view).

The destination folder for the device top view is specified in the Save As dialog box which opens by selecting [File] menu >> [Save Top View As ...].

Figure 2-13. Output Device Top View



Remark If a device top view has been already output, that view will be overwritten by selecting [File] menu >> [Save Top View].

CHAPTER 3 FUNCTIONS (Code Generator)

This chapter describes the key functions provided by the design tool (Code Generator) along with operation procedures.

3.1 Overview

The Code Generator outputs source code (device driver programs) based on information selected/entered on CubeSuite+ panels that is needed to control peripheral functions provided by the microcontroller (clock generator, port functions, etc.).

The following sections describe the operation procedures for Code Generator.

(1) Start CubeSuite+

Launch CubeSuite+ from the [Start] menu of Windows.

Remark See "CubeSuite+ Integrated Development Environment User's Manual: Start" for details on "Start CubeSuite+".

(2) Create/Open project

Create a new project (that defines a kind of project, microcontroller to be used, build tools to be used, etc.) or load an existing project.

Remark See "CubeSuite+ Integrated Development Environment User's Manual: Start" for details on "Create/Open project".

(3) Open Code Generator Panel

Open the Code Generator panel used to configure the information necessary to control the peripheral functions (clock generator, port functions, etc.).

(4) Enter Information

Configure the information necessary to control the peripheral functions in the Code Generator panel.

(5) Confirm Source Code

Confirm the source code (device driver program) that reflects the information configured in the Code Generator panel.

(6) Output Source Code

Output the source code (device driver program) to the specified folder.

(7) Output Report Files

Output report files (a file containing information configured using Code Generator and a file containing information regarding the source code) to the specified folder.

(8) Save project

Save a project.

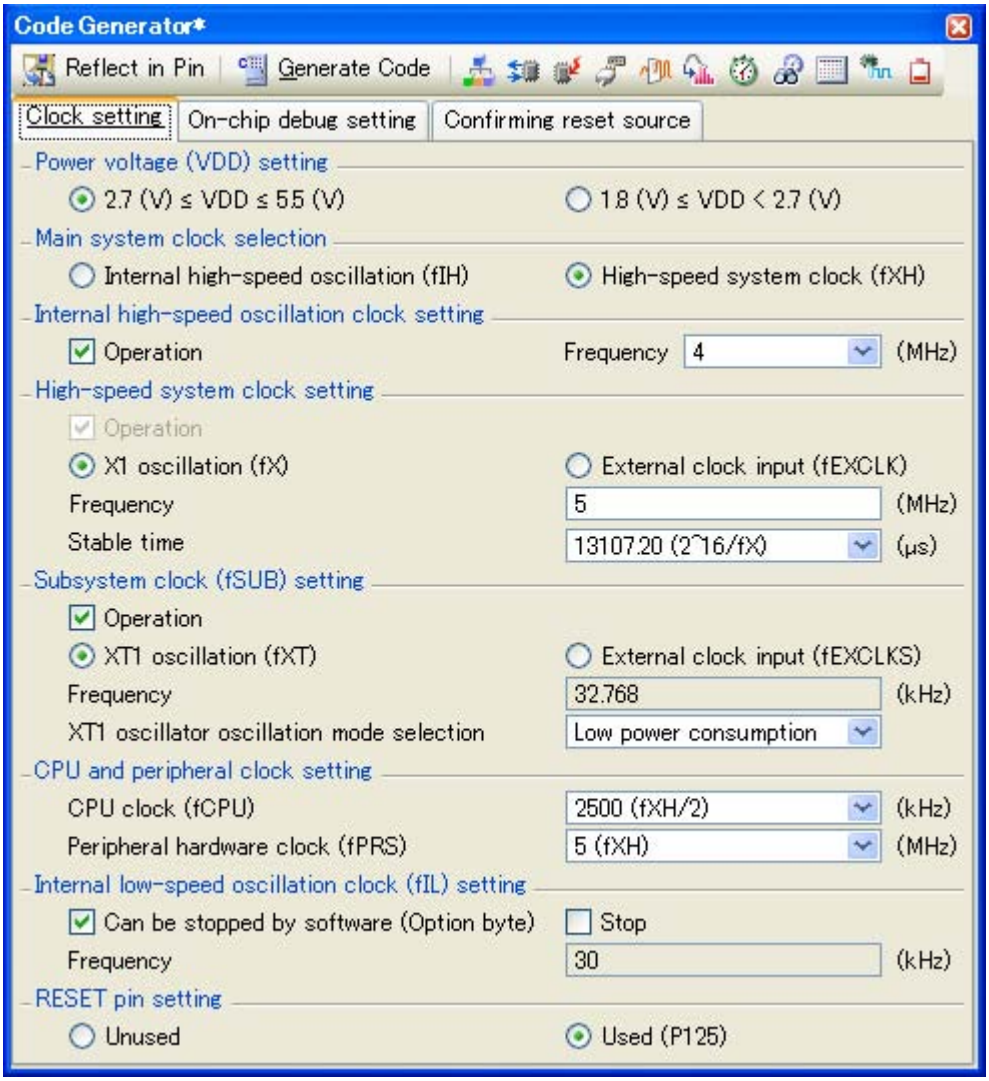
Remark See "CubeSuite+ Integrated Development Environment User's Manual: Start" for details on "Save project".

3.2 Open Code Generator Panel

Open the Code Generator panel to configure the information necessary to control the peripheral functions (clock generator, port functions, etc.).

To open the Code Generator panel, double-click [*Project name* (Project)] >> [Code Generator (Design Tool)] >> Peripheral function node "[System], [Port], etc." in the Project Tree panel.

Figure 3-1. Open Code Generator Panel



Remark If an unsupported microcontroller is defined in the project for Code Generator, then "[Code Generator (Design Tool)] node" will hide under [*Project name* (Project)] in the Project Tree panel.

3.3 Enter Information

Configure the information necessary to control the peripheral functions in the information setting area of the Code Generator panel which is opened as described in "3.2 Open Code Generator Panel".

Remark When controlling multiple peripheral functions, repeat the procedures described in "3.2 Open Code Generator Panel" through "3.3 Enter Information".

3.3.1 Input rule

Following is the rules for input to the Code Generator panel.

(1) Character set

Character sets that are allowed to input are as follows.

Table 3-1. List of Character Set

Character Set	Outline
ASCII	1-byte alphabet, number, symbol
Shift-JIS	2-byte alphabet, number, symbol, Hiragana, Katakana, Kanji and 1-byte Katakana
EUC-JP	2-byte alphabet, number, symbol, Hiragana, Katakana, Kanji and 1-byte Katakana
UTF-8	2-byte alphabet, number, symbol, Hiragana, Katakana, Kanji (include Chinese character) and 1-byte Katakana

(2) Number

Notations allowed when entering numbers are as follows.

Table 3-2. List of Notation

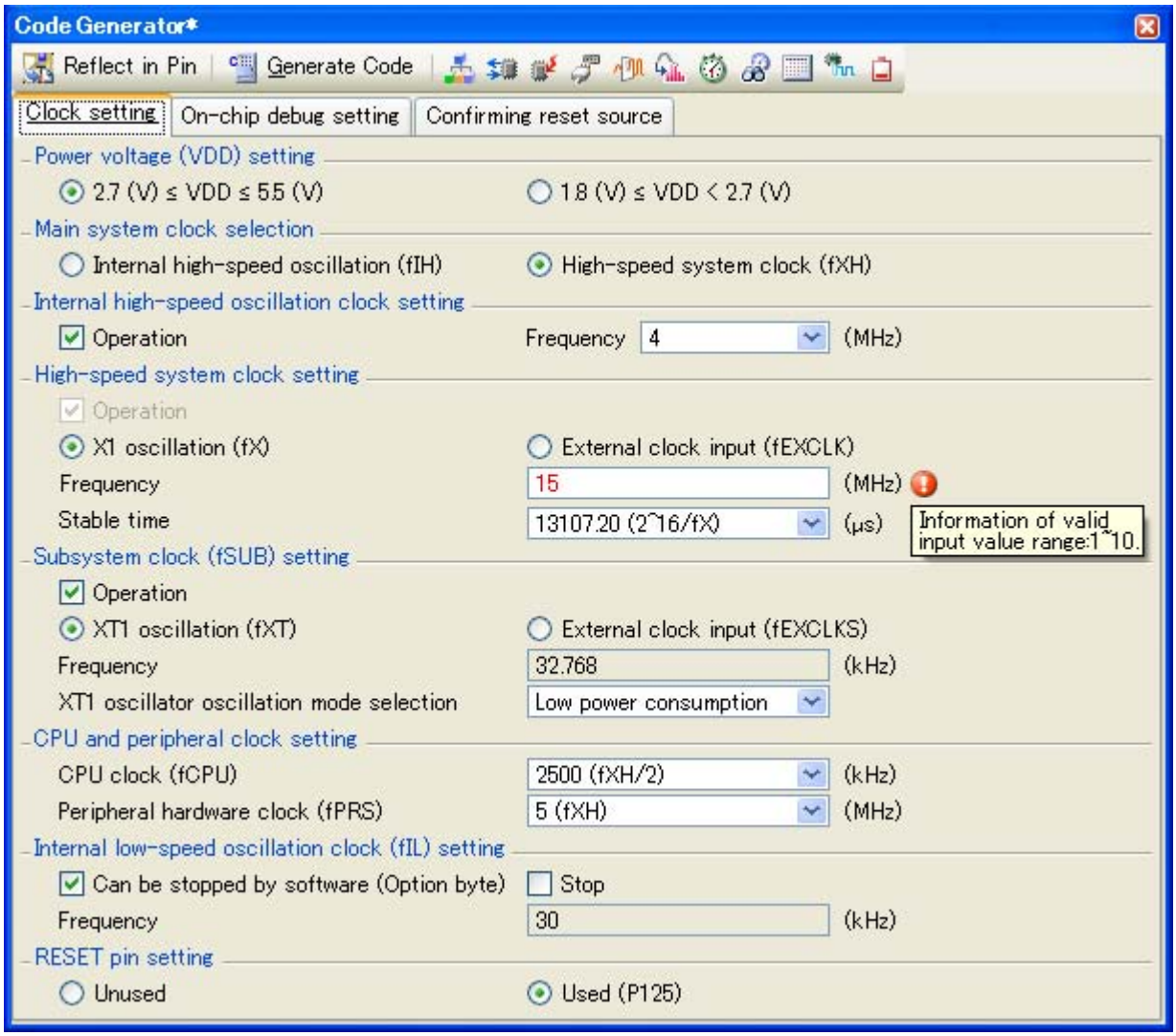
Notation	Outline
Decimal number	A numeric value that starts with a number between 1 and 9 and followed by numbers between 0 and 9, and the numeric value 0
Hex number	A numeric value that starts with 0x and followed by a combination of numbers from 0 to 9 and characters from A to F (characters are not case sensitive)

3.3.2 Icon indicating incorrect entry

When performing code generation, if you enter an invalid string in the Code Generator panel, or a required input is missing, then a icon displays next to the incorrect input, and the text is displayed in red to warn that there is a problem with the input.

Remark If the mouse cursor is moved over the icon, information regarding the string that should be entered (tips for correcting the entry) popups.

Figure 3-2. Icon Indicating Incorrect Entry

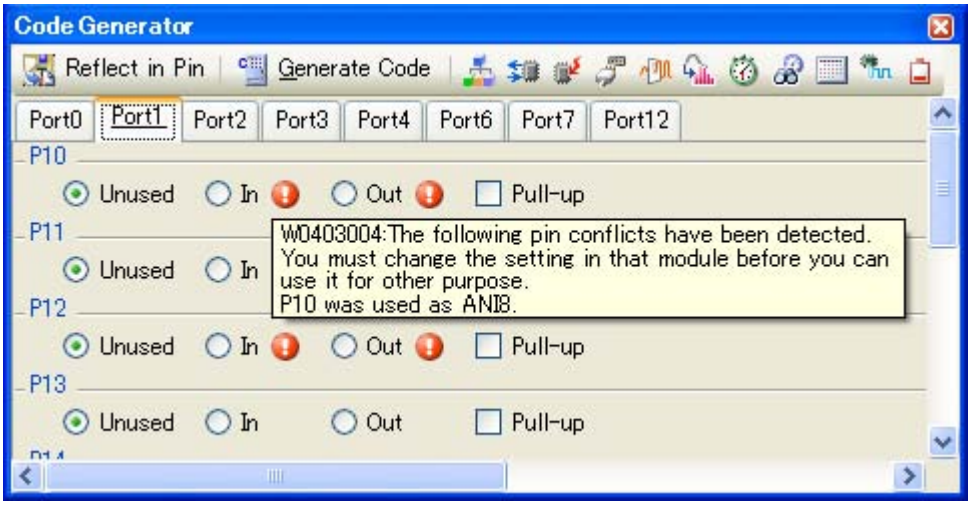


3.3.3 Icon indicating pin conflict

If a conflict occurs between the pins while setting various peripheral functions in the Code Generator panel, the icon is displayed at the location where the conflict occurs to warn the user of a conflict between the pins.

Remark If the mouse cursor is moved over the icon, information regarding the conflict between the pins (tips for avoiding the conflict) popups.

Figure 3-3. Icon Indicating Pin Conflict

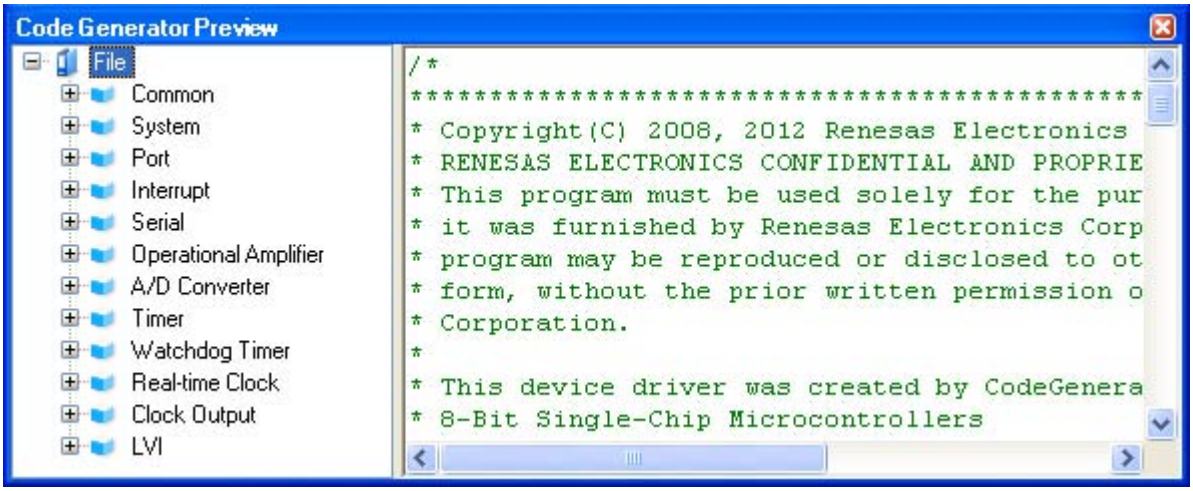


3.4 Confirm Source Code

Confirm the source code (device driver program) that reflects the information configured as described in "3.3 Enter Information".

To confirm the source code, use the Code Generator Preview panel that opens by selecting [View] menu >> [Code Generator Preview].

Figure 3-4. Confirm Source Code



Remarks 1. You can change the source code to be displayed by selecting the source file name or API function name in the Code Generator Preview panel.

2. The following table displays the meaning of the color of the source code text displayed in the Code Generator Preview panel.

Table 3-3. Color of Source Code

Color	Outline
Green	Comment
Blue	Reserved word for C compiler
Red	Numeric value
Black	Code section
Gray	File name

3. You cannot edit the source code within the Code Generator Preview panel.

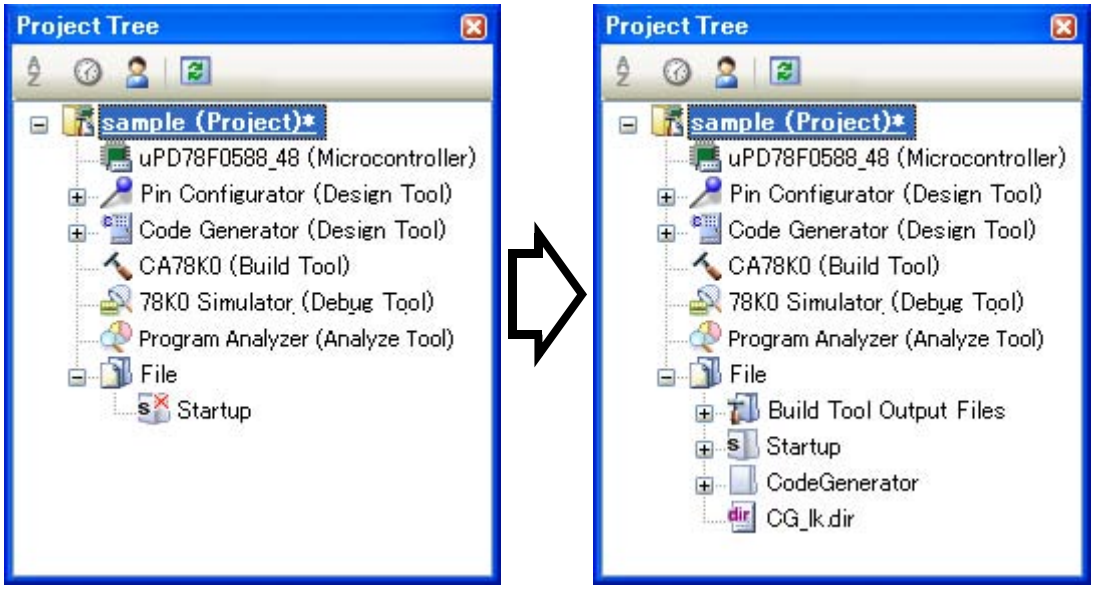
4. For some of the API functions (such as API functions for serial array units), values such as the SFR register value are calculated and finalized when the source code is generated (when the Generate Code button on the Code Generator panel is pressed). For this reason, the source code displayed in the Code Generator Preview panel may not be the same as that would actually be generated.

3.5 Output Source Code

Output the source code (device driver program) by pressing the Generate Code button on the Code Generator panel.

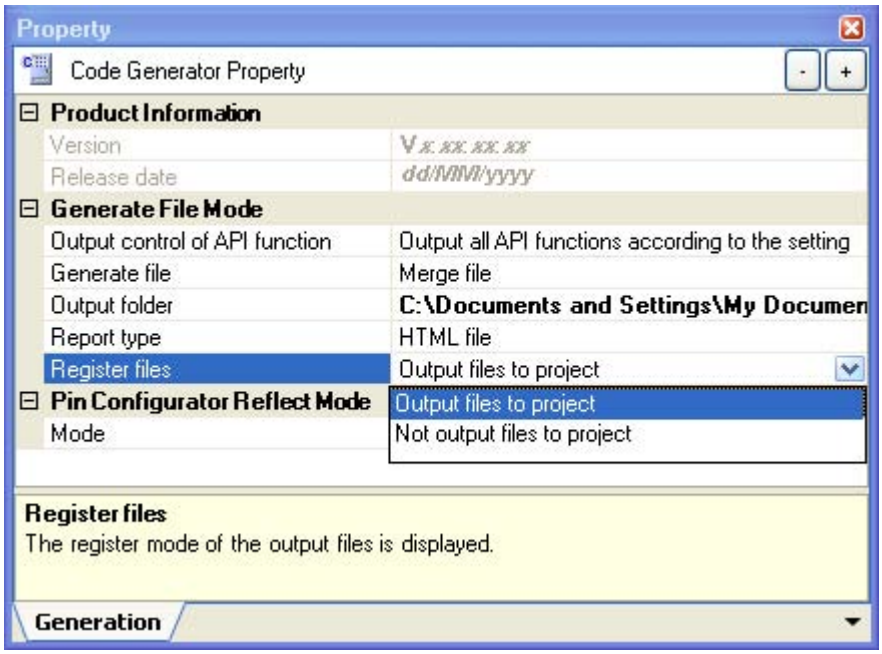
The destination folder for the source code is specified by clicking [Generation] tab >> [Output folder] in the Property panel.

Figure 3-5. Output Source Code



Remark In order to both output source files and add them to the project (display the corresponding source file names in the Project Tree panel) when you click the Generate Code button, you must open the Property panel, and under [Generation] tab >> [Register files], specify "Output files to project".

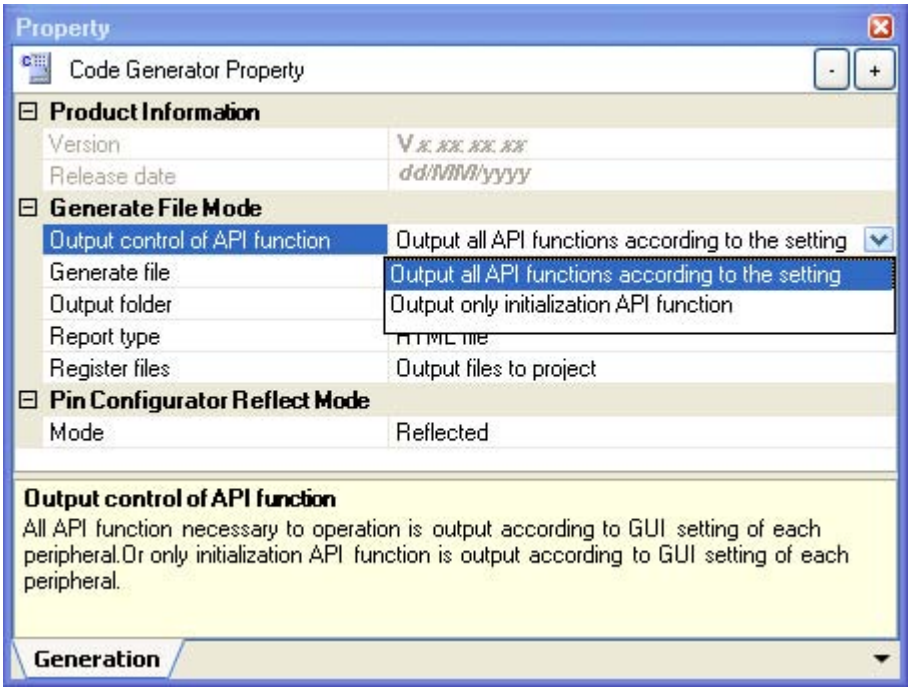
Figure 3-6. Configure Whether to Register



3.5.1 Setting that determines whether or not to generate source code

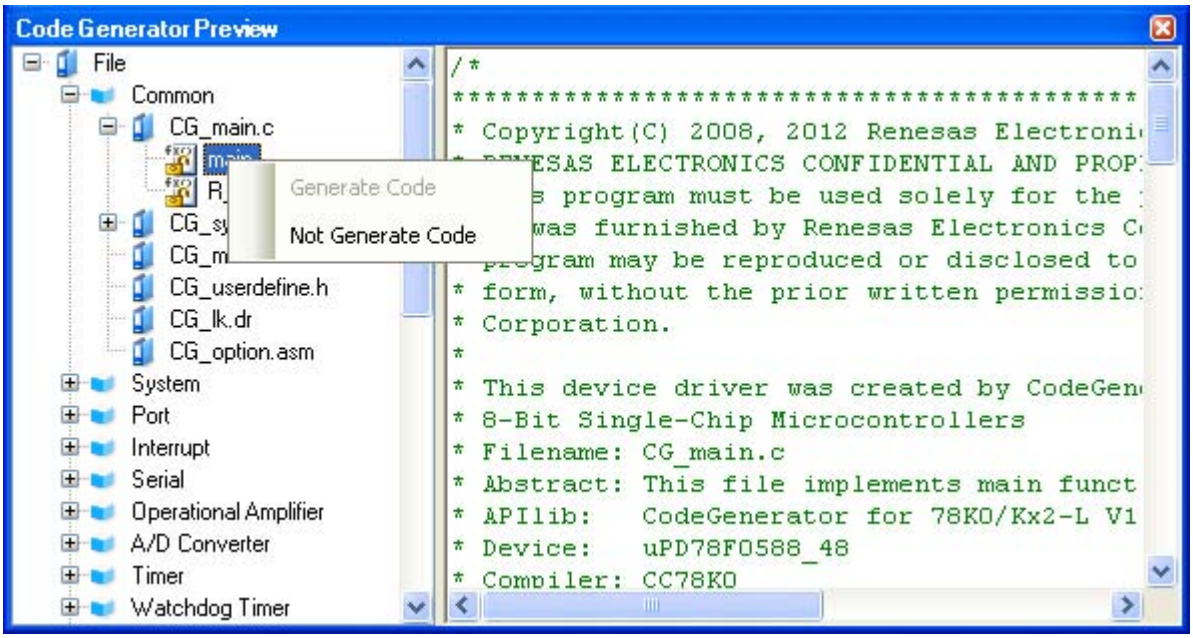
You can set the type of output API functions (all API functions or only initialization API functions) by selecting [Output all API function according to the setting/Output only initialization API function] from [Generation] tab >> [Output control of API function] in the Property panel.

Figure 3-7. Setting That Determines Type of API Functions



You can set whether or not to generate the corresponding source code on a per-API function basis by selecting [Generate code/Not generate code] from the context menu displayed by right clicking the API function name in the Code Generator Preview panel.

Figure 3-8. Setting That Determines Whether or Not to Generate Source Code



Remark You can confirm the current setting for the generation of source code by checking the type of icon in the Code Generator Preview panel.

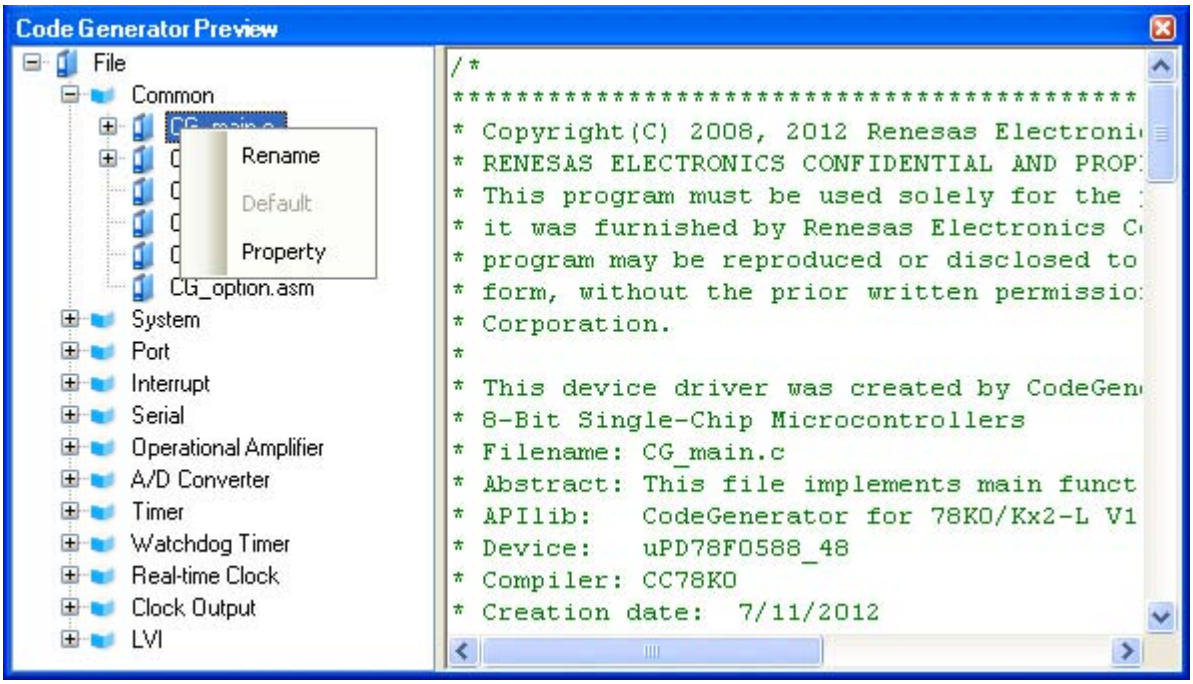
Table 3-4. Setting That Determines Whether or Not to Generate Source Code

Type of Icon	Outline
	Source code for the currently selected API function is generated. If this icon is displayed next to the API function, the corresponding source code must be generated (it is impossible to change the icon to ).
	Source code for the currently selected API function is generated.
	Source code for the currently selected API function is not generated.

3.5.2 Change file name

The Code Generator is used to change the file name by selecting [Rename] from the context menu displayed by right clicking the file name in the Code Generator Preview panel.

Figure 3-9. Change File Name

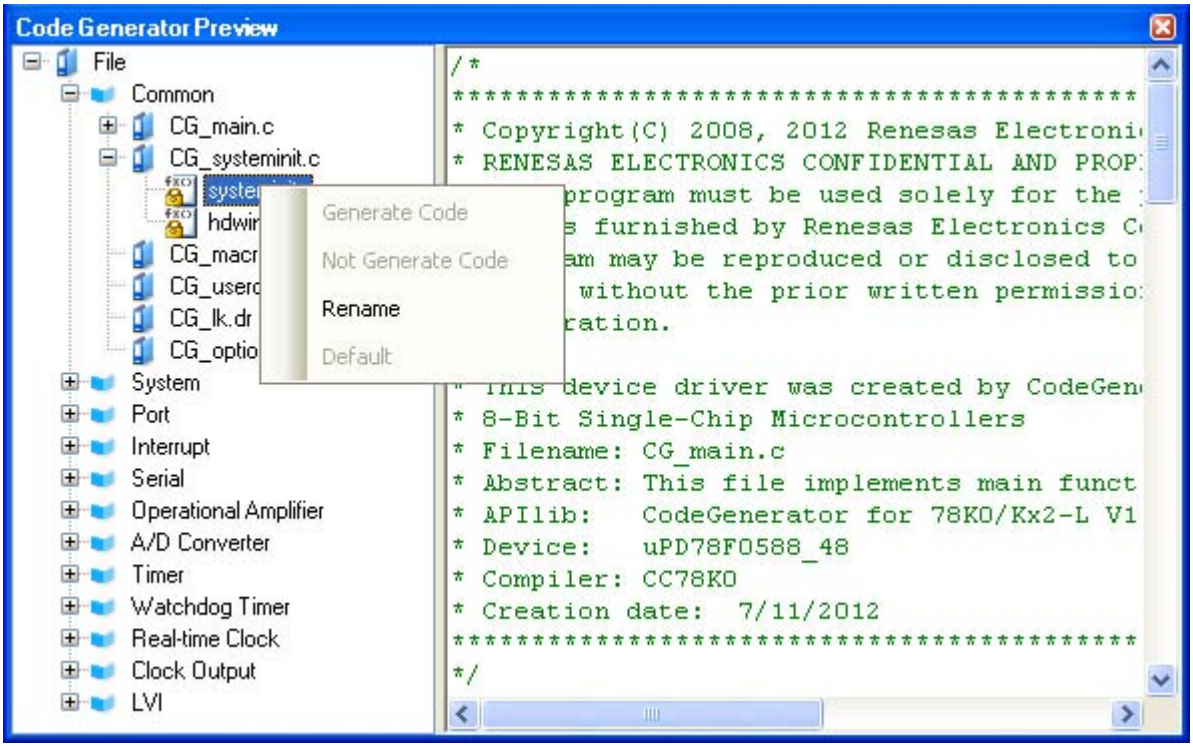


Remark To restore the default file name defined by Code Generator, select [Default] from the context menu.

3.5.3 Change API function name

The Code Generator is used to change the name of the API function by selecting [Rename] from the context menu displayed by right clicking the API function name in the Code Generator Preview panel.

Figure 3-10. Change API Function Name

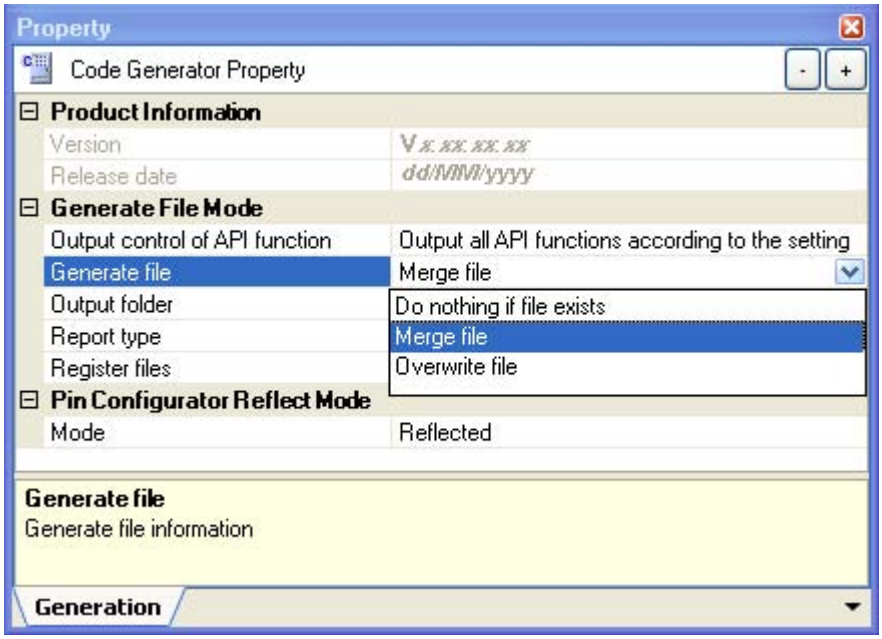


Remark To restore the default name of the API function defined by Code Generator, select [Default] from the context menu.

3.5.4 Change output mode

The Code Generator is used to change the output mode (Do nothing if file exists, Merge file, Overwrite file) for the source code by selecting [Generation] tab >> [Generate file] in the Property panel.

Figure 3-11. Change Output Mode



Remark The output mode is selected from the following three types.

Table 3-5. Output Mode of Source Code

Output Mode	Outline
Do nothing if file exists	If a file with the same name exists, a new file will not be output.
Merge file	If a file with the same name exists, a new file is merged with the existing file. Only the section between "/* Start user code ... . Do not edit comment generated here */" and "/* End user code. Do not edit comment generated here */" will be merged.
Overwrite file	If a file with the same name exists, the existing file is overwritten by a new file.

3.5.5 Change output destination folder

The Code Generator is used to change the output destination folder for the source code by selecting [Generation] tab >> [Output folder] in the Property panel.

To change the output destination, use the Browse For Folder dialog box which opens by pressing the [...] button in the [Output folder].

Figure 3-12. Change Output Destination Folder

3.6 Output Report Files

Output report files (a file containing information configured using Code Generator and a file containing information regarding the source code) by first activating the Code Generator panel or Code Generator Preview panel, then selecting [File] menu >> [Save Code Generator Report].

The destination folder for the report file is specified by clicking [Generation] tab >> [Output folder] in the Property panel.

Remarks 1. You can only use "macro" or "function" as a name of the report file.

Table 3-6. Output Report Files

File Name	Outline
macro	A file that contains the information configured using Code Generator
function	A file that contains the information regarding the source code

2. The output mode for the report file is fixed to "Overwrite file".

Figure 3-13. Output Example of Report File "macro"

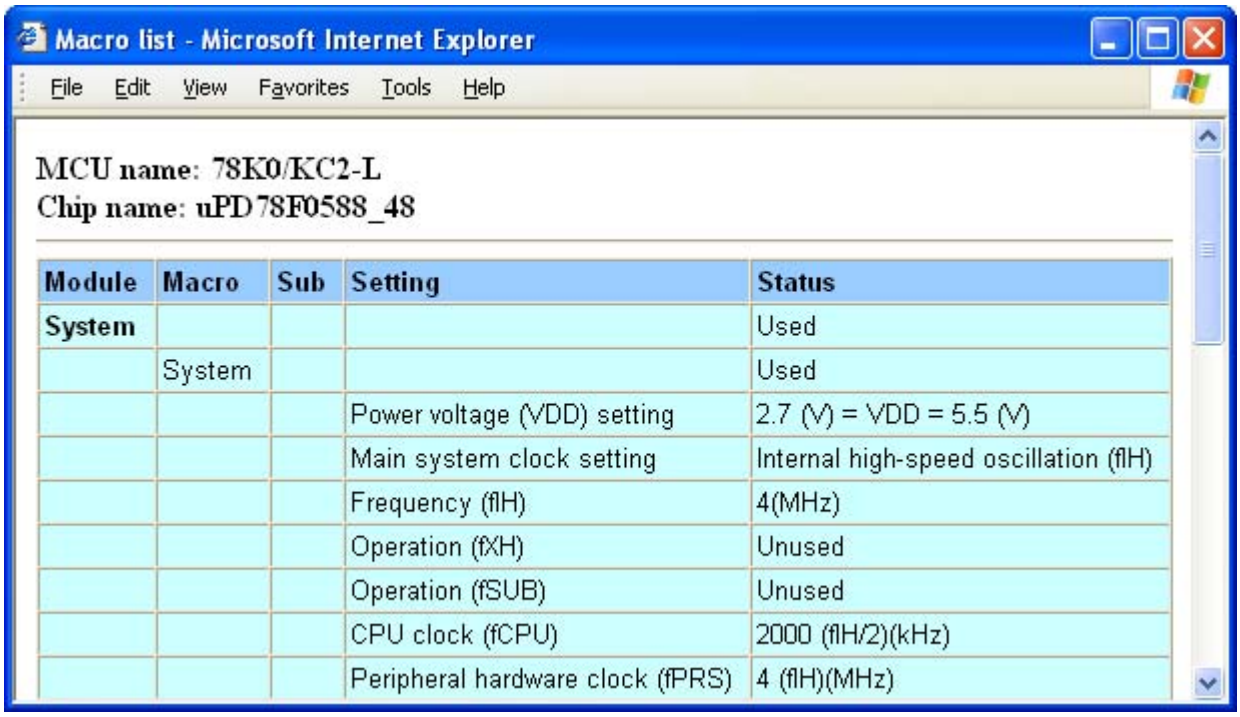
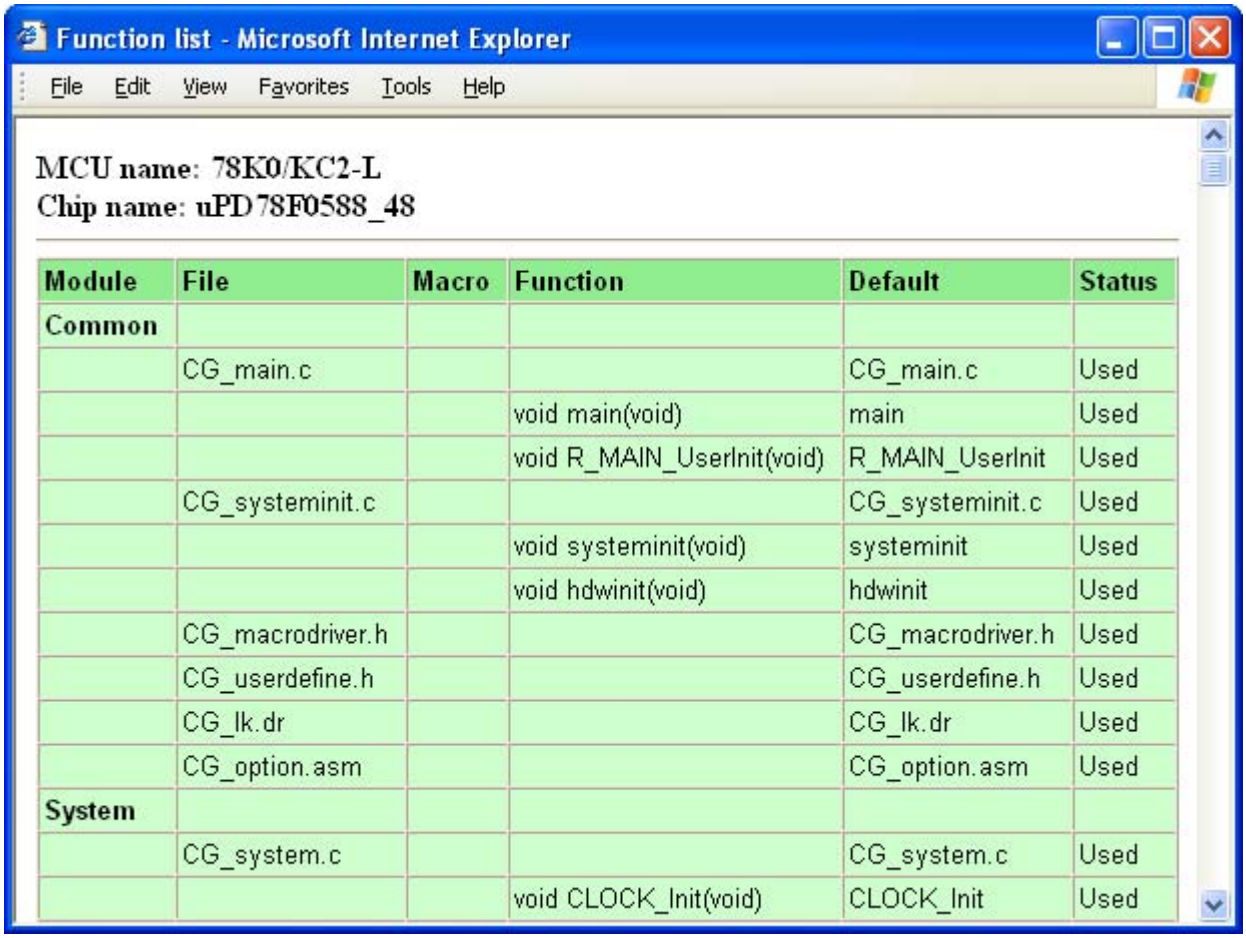


Figure 3-14. Output Example of Report File "function"



Function list - Microsoft Internet Explorer

File Edit View Favorites Tools Help

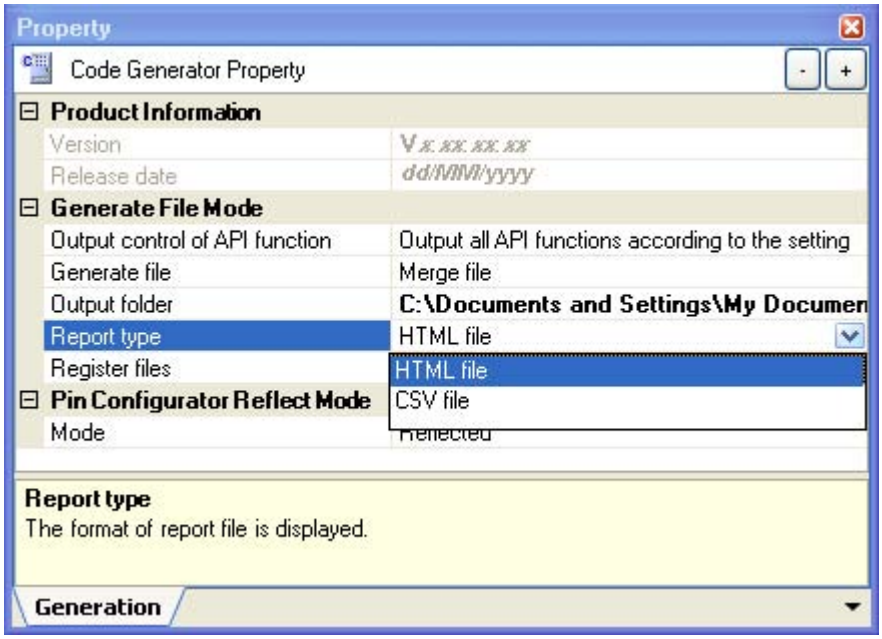
MCU name: 78K0/KC2-L
Chip name: uPD78F0588_48

Module	File	Macro	Function	Default	Status
Common					
	CG_main.c			CG_main.c	Used
			void main(void)	main	Used
			void R_MAIN_UserInit(void)	R_MAIN_UserInit	Used
	CG_systeminit.c			CG_systeminit.c	Used
			void systeminit(void)	systeminit	Used
			void hdwinit(void)	hdwinit	Used
	CG_macrodriver.h			CG_macrodriver.h	Used
	CG_userdefine.h			CG_userdefine.h	Used
	CG_lk.dr			CG_lk.dr	Used
	CG_option.asm			CG_option.asm	Used
System					
	CG_system.c			CG_system.c	Used
			void CLOCK_Init(void)	CLOCK_Init	Used

3.6.1 Change output format

The Code Generator is used to change the output format (HTML file or CSV file) of the report file by selecting [Generation] tab >> [Report type] in the Property panel.

Figure 3-15. Change Output Format



Remark Output format is selected from the following two types.

Table 3-7. Output Mode of Source Code

Report Type	Outline
HTML file	Outputs a report file in HTML format.
CSV file	Outputs a report file in CSV format.

3.6.2 Change output destination

The Code Generator is used to change the output destination folder for the report file by selecting [Generation] tab >> [Output folder] in the Property panel.

To change the output destination, use the Browse For Folder dialog box which opens by pressing the [...] button in the [Output folder].

Figure 3-16. Change Output Destination

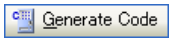
APPENDIX A WINDOW REFERENCE

This appendix explains in detail the functions of the windows, panels and dialog boxes of the design tool.

A.1 Description

The design tool has the following windows, panels and dialog boxes.

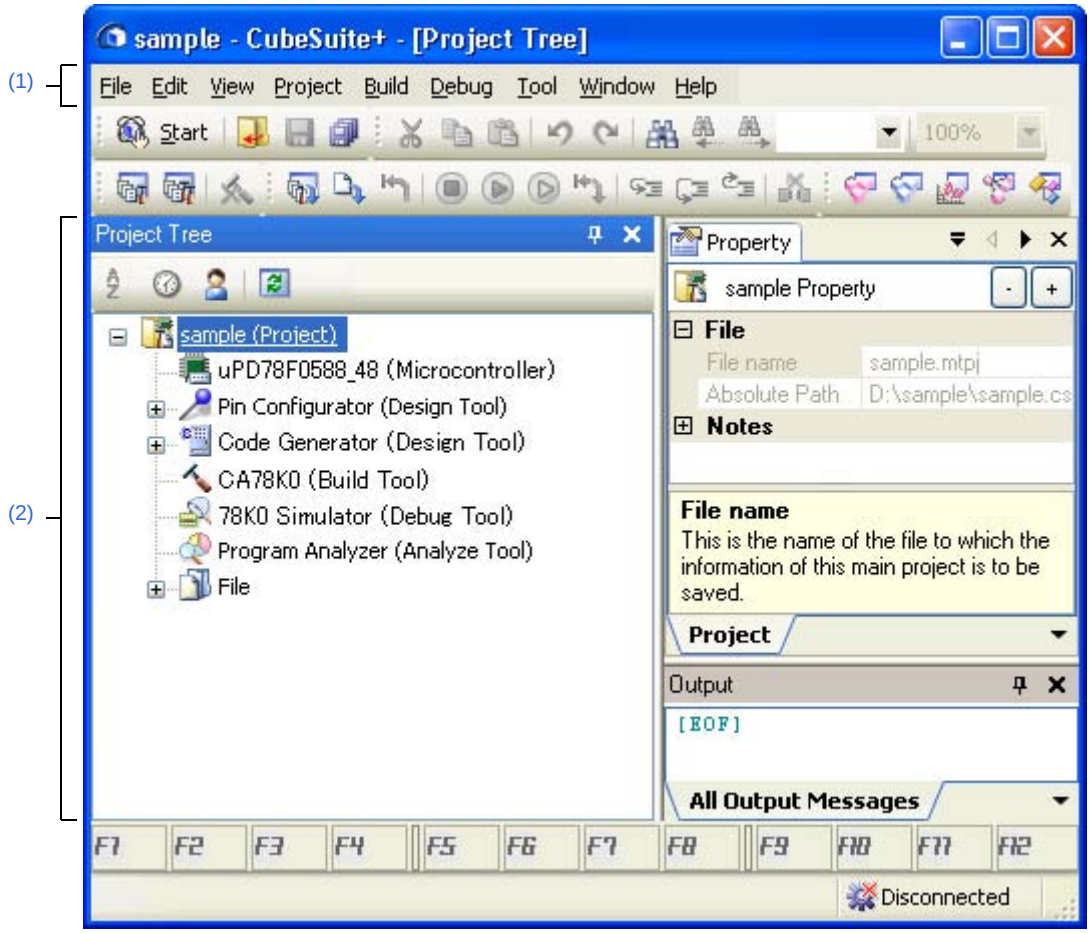
Table A-1. Window/Panel/Dialog Box List

Window/Panel/Dialog Box Name	Function
Main window	This is the first window to open when CubeSuite+ is launched. This window is used to operate various components (design tool, build tool, etc.) provided by CubeSuite+.
Project Tree panel	This panel displays the components of the project (microcontroller, design tool, build tool, etc.) in a tree structure.
Property panel	This panel allows you to view the information and change the setting for the node selected in the Project Tree panel, the peripheral function button pressed in the Code Generator panel or the file selected in the Code Generator Preview panel.
Device Pin List panel	This panel allows you to enter information on each pin of the microcontroller.
Device Top View panel	This panel displays the information entered in the Device Pin List panel.
Code Generator panel	This panel allows you to configure the information necessary to control the peripheral functions provided by the microcontroller.
Code Generator Preview panel	This panel allows you to confirm or configure on a per-API function basis the setting that determines whether or not the source code (device driver program) is generated when the  button is pressed in the Code Generator panel. It also allows you to confirm the source code that reflects the information configured in the Code Generator panel.
Output panel	This panel displays operation logs for various components (design tool, build tool, etc.) provided by CubeSuite+.
Column Chooser dialog box	This dialog box allows you to choose whether or not to display the item listed in this dialog box in the device pin list, and add columns to or delete columns from the device pin list.
New Column dialog box	This dialog box allows you to add your own column to the device pin list.
Browse For Folder dialog box	This dialog box allows you to specify the output destination for files (source code, report file, etc.).
Save As dialog box	This dialog box allows you to name and save a file (such as a report file).

Main window

This is the first window to open when CubeSuite+ is launched. This window is used to operate various components (design tool, build tool, etc.) provided by CubeSuite+.

Figure A-1. Main Window



The following items are explained here.

- [How to open]
- [Description of each area]

[How to open]

- From the [start] menu, select [All Programs] >> [Renesas Electronics CubeSuite+] >>[CubeSuite+].

[Description of each area]

(1) Menu bar

This area consists of the following menu items.

(a) [File] menu

Save Pin List	Device Pin List panel-dedicated item Saves a report file (a file containing information configured using Pin Configurator: device pin list) overwriting the existing file.
Save Pin List As...	Device Pin List panel-dedicated item Opens the Save As dialog box for naming and saving a report file (a file containing information configured using Pin Configurator: device pin list).
Save Top View	Device Top View panel-dedicated item Saves a report file (a file containing information configured using Pin Configurator: device top view) overwriting the existing file.
Save Top View As...	Device Top View panel-dedicated item Opens the Save As dialog box for naming and saving a report file (a file containing information configured using Pin Configurator: device top view).
Save Code Generator Report	Code Generator panel/Code Generator Preview panel-dedicated item Outputs report files (a file containing information configured using Code Generator and a file containing information regarding the source code). - The output format for the report file (either HTML or CSV) is selected by clicking [Generation] tab >> [Report type] in the Property panel. - The destination folder for the report file is specified by clicking [Generation] tab >> [Output folder] in the Property panel.
Save Output-*Tab Name*	Output panel-dedicated item Saves the message corresponding to the specified tab overwriting the existing file.
Save Output-*Tab Name* As...	Output panel-dedicated item Opens the Save As dialog box for naming and saving the message corresponding to the specified tab.

(b) [Edit] menu

Undo	Property panel-dedicated item Cancels the effect of an edit operation to restore the previous state.
Cut	Property panel-dedicated item Sends the character string or lines selected with range selection to the clipboard and deletes them.
Copy	Property panel/Output panel-dedicated item Sends the character string or lines selected with range selection to the clipboard.
Paste	Property panel-dedicated item Inserts the contents of the clipboard at the caret position.
Delete	Property panel-dedicated item Deletes the character string or the lines selected with the range selection.
Select All	Property panel/Output panel-dedicated item Selects all the strings displayed in the item being edited or all the strings displayed in the Message area.

Search...	Device Pin List panel/Code Generator Preview panel/Output panel-dedicated item Opens the Search and Replace dialog box for searching strings with the [Quick Search] tab selected.
Replace...	Output panel-dedicated item Opens the Search and Replace dialog box for replacing strings with the [Whole Replace] tab selected.

(c) [Help] menu

Open Help for Project Tree Panel	Project Tree panel-dedicated item Displays the help of Project Tree panel.
Open Help for Property Panel	Property panel-dedicated item Displays the help of Property panel.
Open Help for Device Pin List Panel	Device Pin List panel-dedicated item Displays the help of Device Pin List panel.
Open Help for Device Top View Panel	Device Top View panel-dedicated item Displays the help of Device Top View panel.
Open Help for Code Generator Panel	Code Generator panel-dedicated item Displays the help of Code Generator panel.
Open Help for Code Generator Preview Panel	Code Generator Preview panel-dedicated item Displays the help of Code Generator Preview panel.
Open Help for Output Panel	Output panel-dedicated item Displays the help of Output panel.

(2) Panel display area

This area consists of multiple panels, each dedicated to a different purpose.

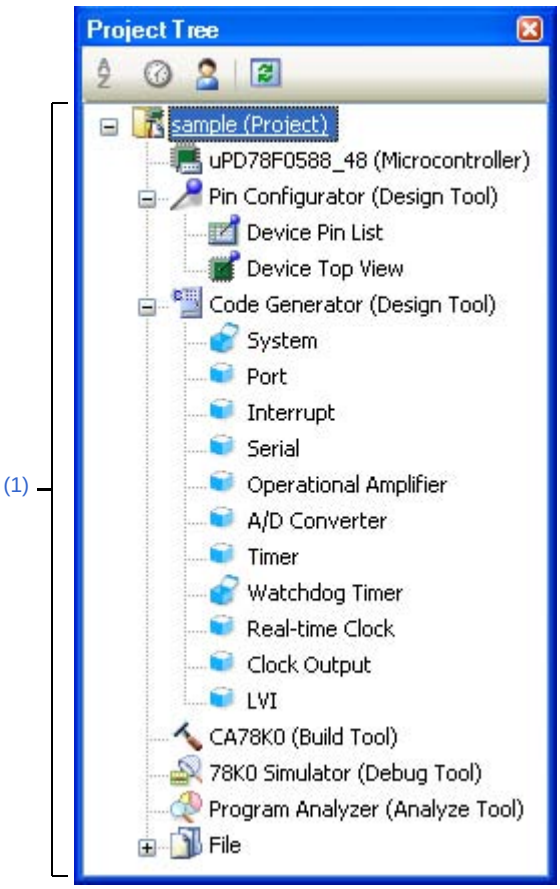
See the following sections for details on this area.

- Project Tree panel
- Property panel
- Device Pin List panel
- Device Top View panel
- Code Generator panel
- Code Generator Preview panel
- Output panel

Project Tree panel

This panel displays components of the project (microcontroller, design tool, build tool, etc.) in a tree structure.

Figure A-2. Project Tree Panel



The following items are explained here.

- [How to open]
- [Description of each area]
- [[Help] menu (Project Tree panel-dedicated items)]
- [Context menu]

[How to open]

- From the [View] menu, select [Project Tree].

[Description of each area]

(1) Project tree area

This area displays components of the project (microcontroller, design tool, build tool, etc.) in a tree structure.

(a) Pin Configurator (Design Tool)

This node consists of the following pin nodes.

Device Pin List	Opens the Device Pin List panel for entering information on the pins of the microcontroller.
Device Top View	Opens the Device Top View panel that displays the information entered in the Device Pin List panel.

(b) Code Generator (Design Tool)

This node consists of the following peripheral function nodes.

When there is peripheral function target microcontroller is not supporting, peripheral functionbutton is not disokayed.

System	Opens the [System] for configuring the information necessary to control the functions of clock generator, on-chip debug function and etc. provided by the microcontroller.
Port	Opens the [Port] for configuring the information necessary to control the port functions provided by the microcontroller.
Interrupt	Opens the [Interrupt] for configuring the information necessary to control the interrupt functions and the key interrupt function provided by the microcontroller.
Serial	Opens the [Serial] for configuring the information necessary to control the functions of serial array unit and functions of serial interface provided by the microcontroller.
Operational Amplifier	Opens the [Operational Amplifier] for configuring the information necessary to control the functions of comparator/programmable gain amplifier provided by the microcontroller.
Comparator	Opens the [Comparator] for configuring the information necessary to control the functions of comparator provided by the microcontroller.
A/D Converter	Opens the [A/D Converter] for configuring the information necessary to control the function of A/D converter provided by the microcontroller.
Timer	Opens the [Timer] for configuring the information necessary to control the functions of timer array unit provided by the microcontroller.
Watchdog Timer	Opens the [Watchdog Timer] for configuring the information necessary to control the functions of watchdog timer provided by the microcontroller.
Real-time Clock	Opens the [Real-time Clock] for configuring the information necessary to control the functions of real-time counter provided by the microcontroller.
Clock Output	Opens the [Clock Output] for configuring the information necessary to control the functions of clock output controller provided by the microcontroller.
LVI	Opens the [LVI] for configuring the information necessary to control the functions of low-voltage detector provided by the microcontroller.

(c) Icons

The table below displays the meaning of the icon displayed to the left of the string representing the peripheral function node.

	Operation in the corresponding Code Generator panel has been carried out.
	Operation in the corresponding Code Generator panel has not been carried out.

	The problem occurs on the settings became the manipulation to the other peripheral function node influences.

[[Help] menu (Project Tree panel-dedicated items)]

Open Help for Project Tree Panel	Displays the help of this panel.

[Context menu]

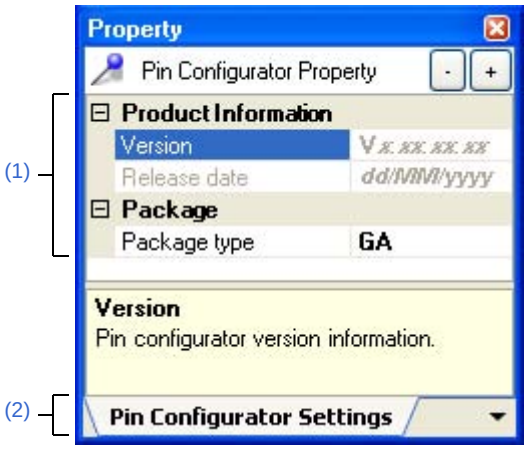
The following context menu items are displayed by right clicking the mouse.

Return to Reset Value	Restores the information for the selected peripheral function node to its default state.
Property	Opens the Property panel containing the information for the selected node ([Pin Configurator (Design Tool)], [Device Pin List], [Device Top View], [Code Generator (Design Tool)], peripheral function node “[System], [Port], etc.”).

Property panel

This panel allows you to view the information on and change the setting for the node selected in the Project Tree panel, the peripheral function button pressed in the Code Generator panel or the file selected in the Code Generator Preview panel.

Figure A-3. Property Panel (Selected [Pin Configurator (Design Tool)])



The following items are explained here.

- [How to open]
- [Description of each area]
- [[Edit] menu (Property panel-dedicated items)]
- [Context menu]

[How to open]

- On the Project Tree panel, select a node ([Pin Configurator (Design Tool)], [Device Pin List], [Device Top View], [Code Generator (Design Tool)], peripheral function node "[System], [Port], etc."), and then select [Property] from the [View] menu.
- On the Project Tree panel, select a node ([Pin Configurator (Design Tool)], [Device Pin List], [Device Top View], [Code Generator (Design Tool)], peripheral function node "[System], [Port], etc."), and then select [Property] from the context menu.
- On the Code Generator Preview panel, select a file, and then select [Property] from the [View] menu.
- On the Code Generator Preview panel, select a file, and then select [Property] from the context menu.

Remarks 1. If this panel is already open, selecting a different node ([Pin Configurator (Design Tool)], [Device Pin List], [Device Top View], [Code Generator (Design Tool)] or peripheral function node "[System], [Port], etc.") in the Project Tree panel changes the content displayed in the Detail information display/change area and explanation area accordingly.

2. If this panel is already open, pressing a different peripheral function button " , , etc." in the Code Generator panel changes the content displayed in the Detail information display/change area and explanation area accordingly.

3. If this panel is already open, selecting a different file in the Code Generator Preview panel changes the content displayed in the Detail information display/change area and explanation area accordingly.

[Description of each area]

(1) Detail information display/change area

This area allows you to view the information on and change the setting for the node ([Pin Configurator (Design Tool)], [Device Pin List], [Device Top View], [Code Generator (Design Tool)] or peripheral function node "[System], [Port], etc.") selected in the Project Tree panel, the peripheral function button "  ,  , etc." pressed in the Code Generator panel, or the file selected in the Code Generator Preview panel.
The content displayed in this area differs depending on the node selected in the Project Tree panel, the peripheral function button pressed in the Code Generator panel or the file selected in the Code Generator Preview panel.

The following table displays the meaning of ⊞ and ⊟ displayed to the left of each category.

⊞	Indicates that the items within the category are displayed as a "collapsed view".
⊟	Indicates that the items within the category are displayed as an "expanded view".

Remark To switch between ⊞ and ⊟ , click this mark or double-click the category name.

(2) Tab selection area

Categories for the display of the detailed information are changed when each tab is selected.
In this panel, following tabs are contained (see the section explaining each tab for details on the display/setting on the tab).

- [Pin Configurator Settings] tab
- [Device Pin List Information] tab
- [Device Top View Settings] tab
- [Generation] tab
- [Macro Setting] tab
- [File Setting] tab

[[Edit] menu (Property panel-dedicated items)]

Undo	Cancels the effect of an edit operation to restore the previous state.
Cut	Sends the character string or lines selected with range selection to the clipboard and deletes them.
Copy	Sends the character string or lines selected with range selection to the clipboard.
Paste	Inserts the contents of the clipboard at the caret position.
Delete	Deletes the character string or the lines selected with the range selection.
Select All	Selects all strings displayed in the item being edited.

[Context menu]

The following context menu items are displayed by right clicking the mouse.

(1) While the item is being edited

Undo	Cancels the effect of an edit operation to restore the previous state.
Cut	Sends the character string or lines selected with range selection to the clipboard and deletes them.

Copy	Sends the character string or lines selected with range selection to the clipboard.
Paste	Inserts the contents of the clipboard at the caret position.
Delete	Deletes the character string or the lines selected with the range selection.
Select All	Selects all strings displayed in the item being edited.

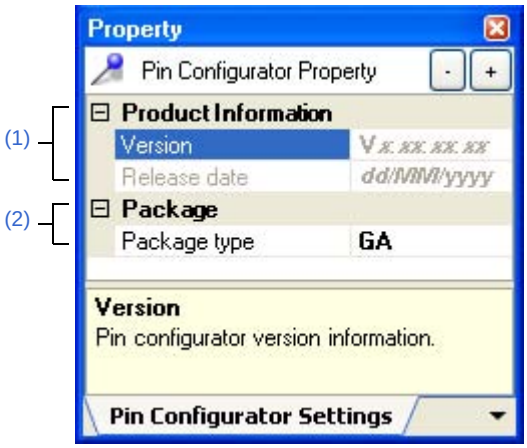
(2) While the item is not being edited

Property Reset to Default	Restores the selected item to its default state.
Property Reset All to Default	Restores all items to their default state.

[Pin Configurator Settings] tab

This tab displays information (Product Information and Package) on the [Pin Configurator (Design Tool)] selected in the Project Tree panel.

Figure A-4. [Pin Configurator Settings] Tab



The following items are explained here.

- [How to open]
- [Description of each area]

[How to open]

- On the Project Tree panel, select [*Project name* (Project)] >> [Pin Configurator (Design Tool)], and then select [Property] from the [View] menu.
- On the Project Tree panel, select [*Project name* (Project)] >> [Pin Configurator (Design Tool)], and then select [Property] from the context menu.

Remark If this panel is already open, selecting a different [Pin Configurator (Design Tool)] in the Project Tree panel changes the content displayed accordingly.

[Description of each area]

(1) [Product Information] category

This area displays product information (Version and Release date) on Pin Configurator.

Version	Displays the version of Pin Configurator (Pin Configurator Plug-in).
Release date	Displays the release date of Pin Configurator (Pin Configurator Plug-in).

(2) [Package] category

Change the shape (Package type) and settings of the microcontroller to display as the device top view in the Device Top View panel.

Package type	Selects the shape of the microcontroller displayed in the device top view.

[Device Pin List Information] tab

This tab displays information (Product Information) on the [Device Pin List] selected in the Project Tree panel.

Figure A-5. [Device Pin List Information] Tab

The following items are explained here.

- [How to open]
- [Description of each area]

[How to open]

- On the Project Tree panel, select [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Pin List], and then select [Property] from the [View] menu.
- On the Project Tree panel, select [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Pin List], and then select [Property] from the context menu.

Remark If this panel is already open, selecting a different [Device Pin List] in the Project Tree panel changes the content displayed accordingly.

[Description of each area]

(1) [Product Information] category

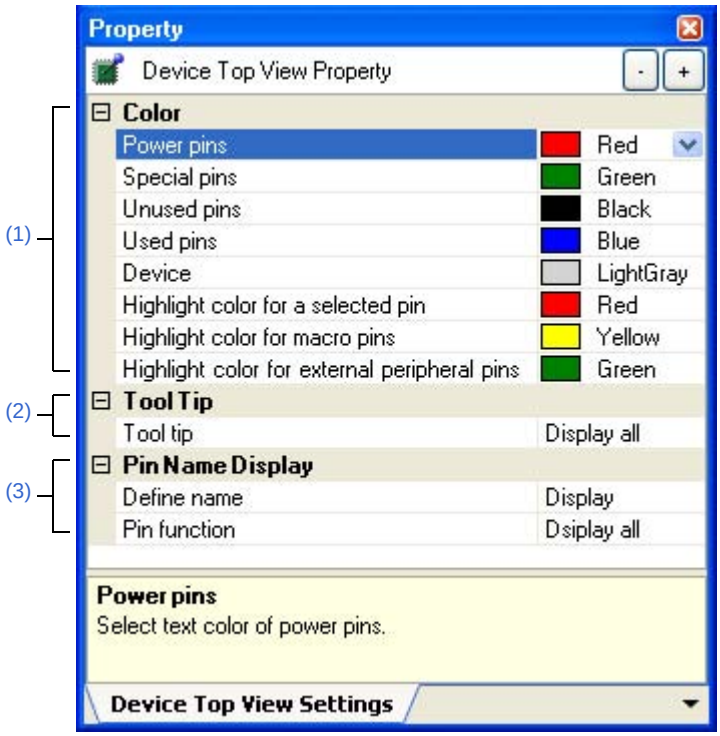
This area displays product information (Version and Release date) on Pin Configurator.

Version	Displays the version of Pin Configurator (Pin Configurator Plug-in).
Release date	Displays the release date of Pin Configurator (Pin Configurator Plug-in).

[Device Top View Settings] tab

This tab allows you to view the information (Color, Tool Tip and Pin Name Display) on and change the setting for the [Device Top View] selected in the Project Tree panel.

Figure A-6. [Device Top View Settings] Tab



The following items are explained here.

- [How to open]
- [Description of each area]

[How to open]

- On the Project Tree panel, select [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Top View], and then select [Property] from the [View] menu.
- On the Project Tree panel, select [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Top View], and then select [Property] from the context menu.

Remark If this panel is already open, selecting a different [Device Top View] in the Project Tree panel changes the content displayed accordingly.

[Description of each area]

(1) [Color] category

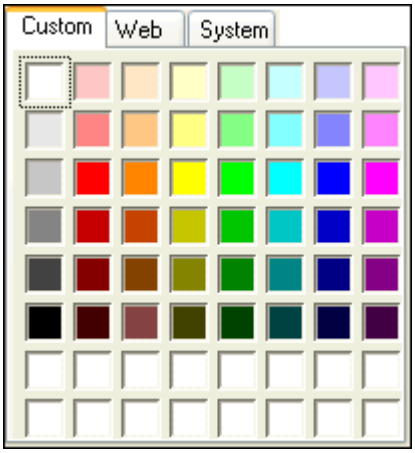
Select the display colors to differentiate the pin groups (Power pins, Special pins, etc.) in the device top view.

Power pins	Selects the display color for power pins (pins whose use is limited to power).
Special pins	Selects the display color for special pins (pins with specified uses).

Unused pins	Selects the display color for unused pins (dual-use pins with no use set in the Device Pin List panel).
Used pins	Selects the display color for used pins (dual-use pins with a use set in the Device Pin List panel).
Device	Selects the display color of the microcontroller.
Highlight color for a selected pin	Selects the background color of a pin selected in the Device Pin List panel, on the [Pin Number] tab.
Highlight color for macro pins	Selects the background color of pins selected in the Device Pin List panel, on the [Macro] tab.
Highlight color for external peripheral pins	Selects the background color of pins selected in the Device Pin List panel, on the [External Peripheral] tab.

Remark To change the setting of the color, use the following color palette which opens by making a selection from the dropdown list in this area.

Figure A-7. Color Palette



(2) [Tool Tip] category

Select whether to display a tooltip with information about a pin when the mouse cursor is moved over the pin in the device top view.

Tool tip	Selects whether to display a tooltip with information about a pin when the mouse cursor is moved over the pin in the device top view panel.	
	Display all	Displays the "Description", "Recommend Connection for Unused", and "Attention" strings for the device pin list.
	Description / recommended connection for unused pin only	Displays the "Description", and "Recommend Connection for Unused" strings for the device pin list.
	Attention only	Displays the "Attention" string for the device pin list.
	Not display	Hides tooltips when the mouse cursor hovers over a pin.

(3) [Pin Name Display] category

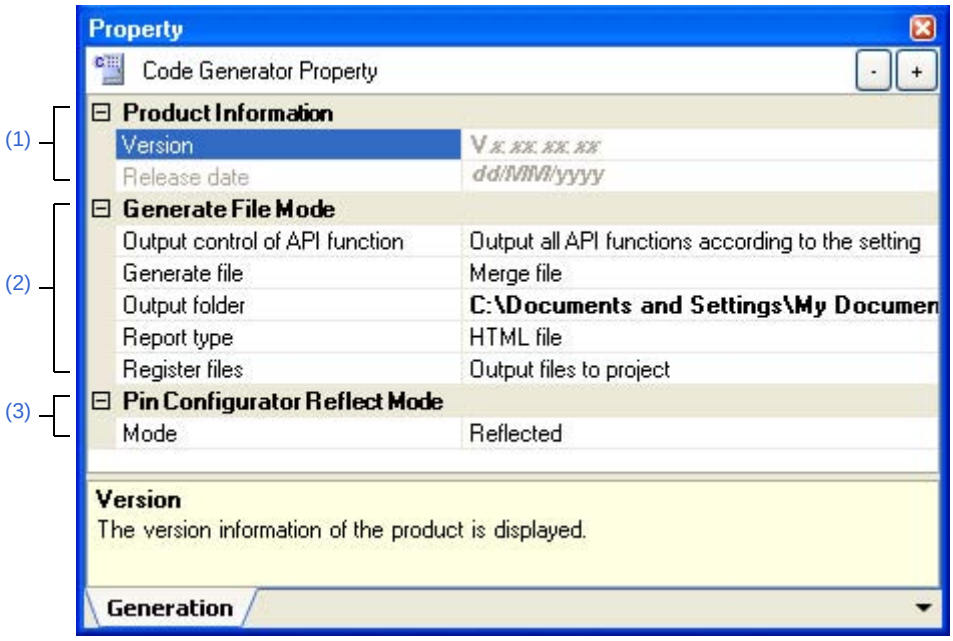
Select whether to display additional information about the pin in the device top view.

Define name	Selects whether to display the "Define Name" string of the device pin list appended to the pin in the device top view.	
	Display	Displays the "Define Name" string of the device pin list in appended format.
	Not display	Hides the "Define Name" string of the device pin list.
Pin function	Selects whether to also display unselected functions in the device top view when a function has been selected from the device pin list's "Function" feature.	
	Display all	Displays functions selected via the device pin list's "Function" feature in parentheses.
	Selected function only	Only display functions selected via the device pin list's "Function" feature in the device top view.

[Generation] tab

This tab allows you to view the information (Product Information, Generate File Mode and Pin Configurator Reflect Mode) on and change the setting for the [Code Generator (Design Tool)] selected in the Project Tree panel.

Figure A-8. [Generation] Tab



The following items are explained here.

- [How to open]
- [Description of each area]

[How to open]

- On the Project Tree panel, select [*Project name* (Project)] >> [Code Generator (Design Tool)], and then select [Property] from the [View] menu.
- On the Project Tree panel, select [*Project name* (Project)] >> [Code Generator (Design Tool)], and then select [Property] from the context menu.

Remark If this panel is already open, selecting a different [Code Generator (Design Tool)] in the Project Tree panel changes the content displayed accordingly.

[Description of each area]

(1) [Product Information] category

This area displays product information (Version and Release date) on Code Generator.

Version	Displays the version of Code Generator (Code Library).
Release date	Displays the release date of Code Generator (Code Library).

(2) [Generate File Mode] category

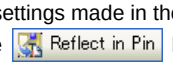
This area allows you to view and change the setting for the file generation mode (Output control of API function, Generate file, etc.) of Code Generator.

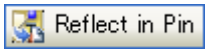
Output control of API function	Views or Selects the type of output API functions (all API functions or only initialization API functions) when the Generate Code button is pressed.	
	Output all API functions according to the setting	Outputs all API functions.
	Output only initialization API function	Outputs only initialization API functions.
Generate file	Views or selects the operation mode applied when the Generate Code button is pressed. Operation mode applied when you select [File] menu >> [Save Code Generator Report] is fixed to "Overwrite file".	
	Do nothing if file exists	If a file with the same name exists, a new file will not be output.
	Merge file	If a file with the same name exists, a new file is merged with the existing file. Only the section between "/* Start user code ... . Do not edit comment generated here */" and "/* End user code. Do not edit comment generated here */" will be merged.
	Overwrite file	If a file with the same name exists, the existing file is overwritten by a new file.
Output folder	Views or selects the destination folder for various files (source code and report files) which are output when the Generate Code button is pressed or when [File] menu >> [Save Code Generator Report] is selected.	
Report type	Views or selects the format of the report files (a file containing information configured using Code Generator and a file containing information regarding the source code) which are output when [File] menu >> [Save Code Generator Report] is selected.	
	HTML file	Outputs a report file in HTML format.
	CSV file	Outputs a report file in CSV format.
Register files	Selects whether source code generated by pressing the Generate Code button should be added to the project.	
	Output files to project	Adds output source code to the project. The source code will be added to the Project Tree panel, under the [File] - [Code Generator] node.
	Not output files to project	Does not add output source code to the project.

Remark To change the output destination, use the Browse For Folder dialog box which opens by pressing the [...] button in this area.

(3) [Pin Configurator Reflect Mode] category

Configure the information linking (Mode) between Code Generator and Pin Configurator.

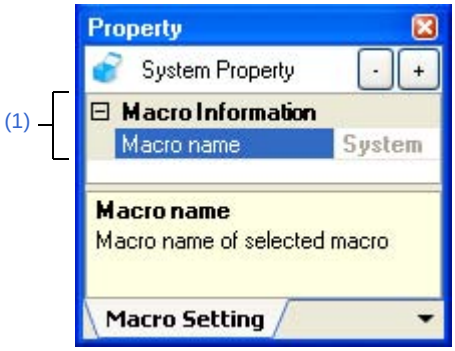
Mode	Selects whether to reflect the settings made in the Code Generator panel in the Device Pin List panel when the  button is pressed.	
	Reflected	Reflects Code Generator panel settings in the Device Pin List panel.
	Not reflected	Does not reflect Code Generator panel settings in the Device Pin List panel.

Remark If "Not reflected" is selected, then the  button will be grayed out (deselected).

[Macro Setting] tab

This tab allows you to view the information (Macro Information) on and change the setting for the peripheral function node "[System], [Port], etc." selected in the Project Tree panel, or the peripheral function button " , , etc." pressed in the Code Generator panel.

Figure A-9. [Macro Setting] Tab



The following items are explained here.

- [How to open]
- [Description of each area]

[How to open]

- On the Project Tree panel, select [*Project name* (Project)] >> [Code Generator (Design Tool)] >> Peripheral function node "[System], [Port], etc.", and then select [Property] from the [View] menu.
- On the Project Tree panel, select [*Project name* (Project)] >> [Code Generator (Design Tool)] >> Peripheral function node "[System], [Port], etc.", and then select [Property] from the context menu.

Remarks 1. If this panel is already open, selecting a different peripheral function node "[System], [Port], etc." in the Project Tree panel changes the content displayed accordingly.

2. If this panel is already open, pressing a different type of peripheral function button " , , etc." in the Code Generator panel changes the content displayed accordingly.

[Description of each area]

(1) [Macro Information] category

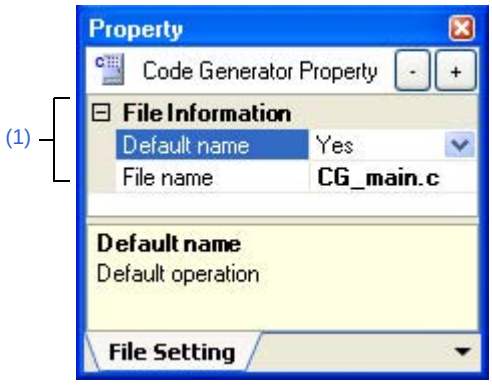
This area allows you to view the information (Macro name) on and change the setting for the peripheral function node "[System], [Port], etc." selected in the Project Tree panel, or the peripheral function button " , , etc." pressed in the Code Generator panel.

Macro name	Displays the type of peripheral function node selected in the Project Tree panel or the type of peripheral function button pressed in the Code Generator panel.

[File Setting] tab

This tab allows you to view the information (File Information) on and change the setting for the file selected in the Code Generator Preview panel.

Figure A-10. [File Setting] Tab



The following items are explained here.

- [How to open]
- [Description of each area]

[How to open]

- On the Code Generator Preview panel, select a file, and then select [Property] from the [View] menu.
- On the Code Generator Preview panel, select a file, and then select [Property] from the context menu.

Remark If this panel is already open, selecting a different file in the Code Generator Preview panel changes the content displayed accordingly.

[Description of each area]

(1) [File Information] category

This area allows you to view the information (Default name and File name) on and change the setting for the file selected in the Code Generator Preview panel.

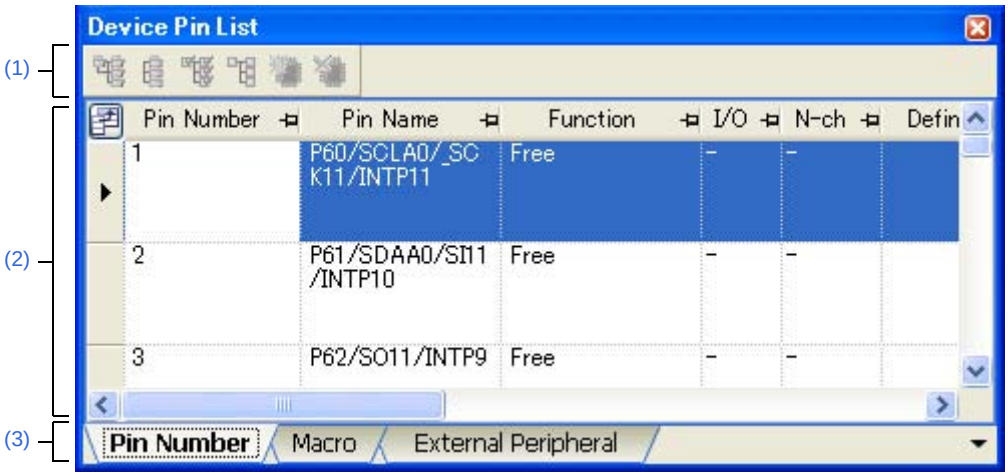
Default name	Views or selects the setting that determines whether the name of the file selected in the Code Generator Preview panel is a default name or not.	
	Yes	The file name is a default name. Changing this area from "No" to "Yes" changes the name of the file to its default name.
	No	The file name is not a default name.
File name	Displays or change the name of the file selected on the Code Generator Preview panel.	

Device Pin List panel

This panel allows you to enter information on each pin of the microcontroller.

Remark The Device pin list area can be zoomed in and out by 100% in the tool bar, or by operating the mouse wheel while holding down the [Ctrl] key.

Figure A-11. Device Pin List Panel



The following items are explained here.

- [How to open]
- [Description of each area]
- [[File] menu (Device Pin List panel-dedicated items)]
- [[Help] menu (Device Pin List panel-dedicated items)]

[How to open]

- On the Project Tree panel, double-click [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Pin List].
- On the Project Tree panel, select [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Pin List], and then press the [Enter] key.
- From the [View] menu, select [Pin Configurator] >> [Device Pin List].

[Description of each area]

(1) Toolbar

This area consists of the following buttons.

	Displays the information in the Device pin list area in an expanded view.
	Displays the information in the Device pin list area in a folded view only.
	Clicks this button to automatically process the configuration information in the selected function, I/O, N-ch, and other fields after selecting one of the peripheral functions displayed in the first level on the [Macro] tab.

	Clicks this button to initialize the selected function, I/O, N-ch, and other fields after selecting one of the peripheral functions displayed in the first level on the [Macro] tab.
	Clicks this button to create an external peripheral controller from the external peripheral controller information on the [External Peripheral] tab, and display it in the Device Top View panel.
	Clicks this button to delete the information for the external peripheral controller displayed on the [External Peripheral] tab, on the first layer.

Remarks 1. Click the  button to add the information in question as a choice in the "External Parts" column of the [Macro] tab and the [Pin Number] tab.

2. Click the  button to remove the external peripheral component in question from the Device top view area if the Device Top View panel.

(2) Device pin list area

This area displays the "device pin list" for entering information on the pins of the microcontroller.

(3) Tab selection area

Selecting the tab changes the order in which "information on each pin of the microcontroller" is displayed.

This panel has the following tabs:

- [Pin Number] tab

  This tab displays information on each pin of the microcontroller in the order of pin number.

- [Macro] tab

  This tab displays information on each pin of the microcontroller in the order it was grouped into peripheral functions.

- [External Peripheral] tab

  This tab displays information about the pins connected to external peripherals in order grouped at the external-peripheral component level.

[[File] menu (Device Pin List panel-dedicated items)]

Save Pin List	Saves a report file (a file containing information configured using Pin Configurator: device pin list) overwriting the existing file.
Save Pin List As...	Opens the Save As dialog box for naming and saving a report file (a file containing information configured using Pin Configurator: device pin list).

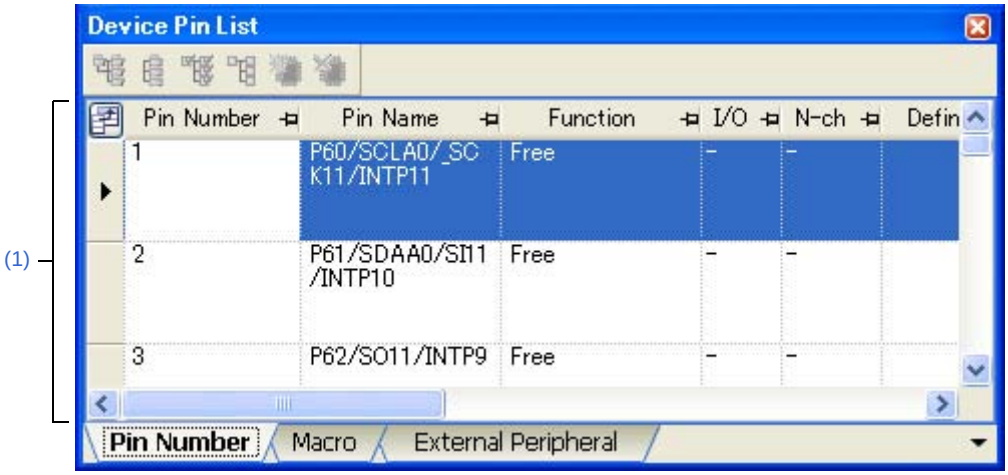
[[Help] menu (Device Pin List panel-dedicated items)]

Open Help for Device Pin List Panel	Displays the help of this panel.

[Pin Number] tab

This tab displays information on each pin of the microcontroller in the order of pin number.

Figure A-12. [Pin Number] Tab



The following items are explained here.

- [How to open]
- [Description of each area]

[How to open]

- On the Project Tree panel, double-click [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Pin List].
- On the Project Tree panel, select [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Pin List], and then press the [Enter] key.
- From the [View] menu, select [Pin Configurator] >> [Device Pin List].

[Description of each area]

(1) Device pin list area

This area displays the "device pin list" for entering information on the pins of the microcontroller.
The device pin list in this area is organized in the order of pin number.
The following are the columns comprising the device pin list.

Column Heading	Outline
Pin Number	Displays the pin number of the pin.
Pin Name	This area allows you to select "which function to use" when the pin has more than one functions.
Function	This area allows you to select "which function to use" when the pin has more than one functions.
I/O	This area allows you to select the I/O mode of the pin.
N-ch	This area allows you to select "which output mode to apply" when using the pin in the output mode.

Column Heading	Outline
Define Name	This area allows you to assign a "user-defined pin name" to the pin. Within 256 characters can be entered in the [Define Name].
Description	Displays the summary of function of the pin.
Recommend Connection for Unused	Displays instructions on how to handle the pin when it is not used. This column displays information only when the "Free" is selected in the "Function" column.
Attention	Displays the precaution on using the pin.
External Parts	This area is for selecting which external peripheral controller to connect the pin to.

Remarks 1. You cannot add information in the "Pin Number" column, "Pin Name" column, "Description" column, "Recommend Connection for Unused" column and "Attention" column because they contain fixed information.

2. If the "Free" in the "Function" column is changed to a specific pin name, color of the corresponding pin in the Device Top View panel changes from the "color representing the unused pins" to the "color representing the used pins" selected by clicking [Device Top View Settings] tab >> [Color] in the Property panel.

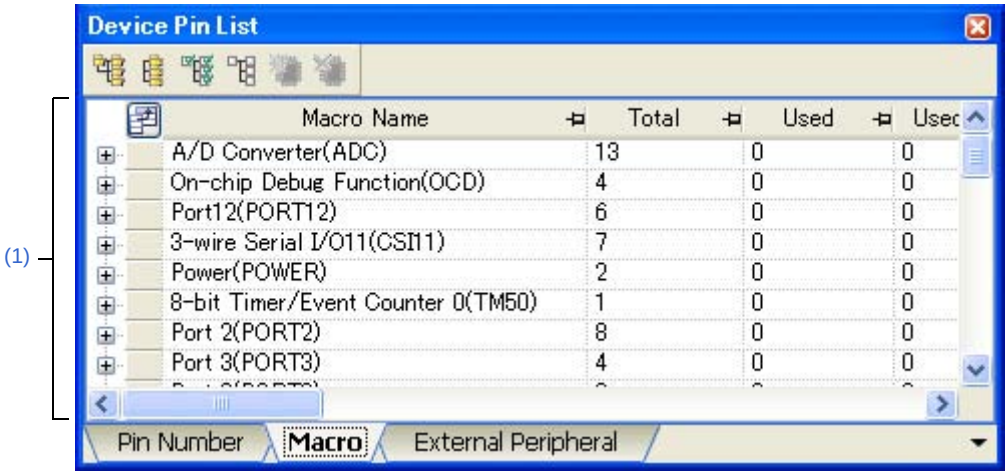
3. To move columns (change the display order) in the device pin list, drag and drop the desired column to the desired location.

4. To add the "user's own column", use the New Column dialog box which opens by pressing the [New Column...] button in the Column Chooser dialog box which opens by pressing the  button in the upper left corner of the device pin list.

[Macro] tab

This tab displays information on each pin of the microcontroller in the order it was grouped into peripheral functions.

Figure A-13. [Macro] Tab



The following items are explained here.

- [How to open]
- [Description of each area]

[How to open]

- On the Project Tree panel, double-click [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Pin List].
- On the Project Tree panel, select [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Pin List], and then press the [Enter] key.
- From the [View] menu, select [Pin Configurator] >> [Device Pin List].

[Description of each area]

(1) Device pin list area

This area displays the "device pin list" for entering information on the pins of the microcontroller.
The device pin list in this area is organized in the order the pins were grouped into peripheral functions.

(a) First layer

The following are the columns comprising the device pin list.

Column Heading	Outline
Macro Name	Displays the name of the peripheral function.
Total	Displays the total number of pins assigned to the peripheral function.
Used	Displays the total number of pins for which the purpose has been set.
Used in Other Macro	Displays the total number of pins for which the purpose has been set by other peripheral functions.

(b) Second layer

Column Heading	Outline
Pin Number	Displays the pin number of the pin.
Pin Name	Displays the pin name of the pin.
Function	This area allows you to select "which function to use" when the pin has more than one functions.
I/O	This area allows you to select the I/O mode of the pin.
N-ch	This area allows you to select "which output mode to apply" when using the pin in the output mode.
Define Name	This area allows you to assign a "user-defined pin name" to the pin. Within 256 characters can be entered in the [Define Name].
Description	Displays the summary of function of the pin.
Recommend Connection for Unused	Displays instructions on how to handle the pin when it is not used. This column displays information only when the "Free" is selected in the "Function" column.
Attention	Displays the precaution on using the pin.
External Parts	This area is for selecting which external peripheral controller to connect the pin to.

Remarks 1. You cannot add information in the "Macro Name", "Total", "Used", "Used by other function", "Pin Number", "Pin Name", "Description", "Recommend Connection for Unused" and "Attention" columns because they contain fixed information.

2. If the "Free" in the "Function" column is changed to a specific pin name, color of the corresponding pin in the Device Top View panel changes from the "color representing the unused pins" to the "color representing the used pins" selected by clicking [Device Top View Settings] tab >> [Color] in the Property panel.

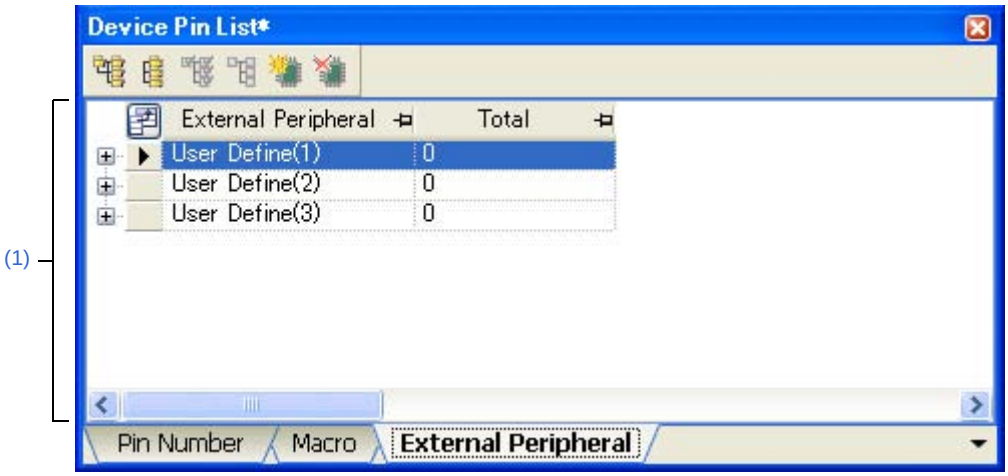
3. To move columns (change the display order) in the device pin list, drag and drop the desired column to the desired location.

4. To add the "user's own column", use the New Column dialog box which opens by pressing the [New Column...] button in the Column Chooser dialog box which opens by pressing the  button in the upper left corner of the device pin list.

[External Peripheral] tab

This tab displays information about the pins connected to external peripherals in order grouped at the external-peripheral component level.

Figure A-14. [External Peripheral] Tab



The following items are explained here.

- [How to open]
- [Description of each area]

[How to open]

- On the Project Tree panel, double-click [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Pin List].
- On the Project Tree panel, select [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Pin List], and then press the [Enter] key.
- From the [View] menu, select [Pin Configurator] >> [Device Pin List].

[Description of each area]

(1) Device pin list area

This area displays the "device pin list" for entering information on the pins connected to external peripheral parts.
Note that items in this area's device pin list are sorted by groups at the external peripheral controller level.

(a) First layer

The following are the columns comprising the device pin list.

Column Heading	Outline
External Peripheral	Displays the name of the external peripheral controller. To change the name, select this field and then press the [F2] key.
Total	Displays the total number of pins allocated for connection with the microcontroller.

(b) Second layer

Column Heading	Outline
Pin Number	Displays the pin number of the pin.
Pin Name	Displays the pin name of the pin.
Function	This area allows you to select "which function to use" when the pin has more than one functions.
I/O	This area allows you to select the I/O mode of the pin.
N-ch	This area allows you to select "which output mode to apply" when using the pin in the output mode.
Define Name	This area allows you to assign a "user-defined pin name" to the pin. Within 256 characters can be entered in the [Define Name].
Description	Displays the summary of function of the pin.
Recommend Connection for Unused	Displays instructions on how to handle the pin when it is not used. This column displays information only when the "Free" is selected in the "Function" column.
Attention	Displays the precaution on using the pin.

Remarks 1. You cannot add information in the "External Peripheral Name", "Connected Pins", "Pin Number", "Pin Name", "Description", "Recommend Connection for Unused" and "Attention" columns because they contain fixed information.

2. If the "Free" in the "Function" column is changed to a specific pin name, color of the corresponding pin in the Device Top View panel changes from the "color representing the unused pins" to the "color representing the used pins" selected by clicking [Device Top View Settings] tab >> [Color] in the Property panel.

3. To move columns (change the display order) in the device pin list, drag and drop the desired column to the desired location.

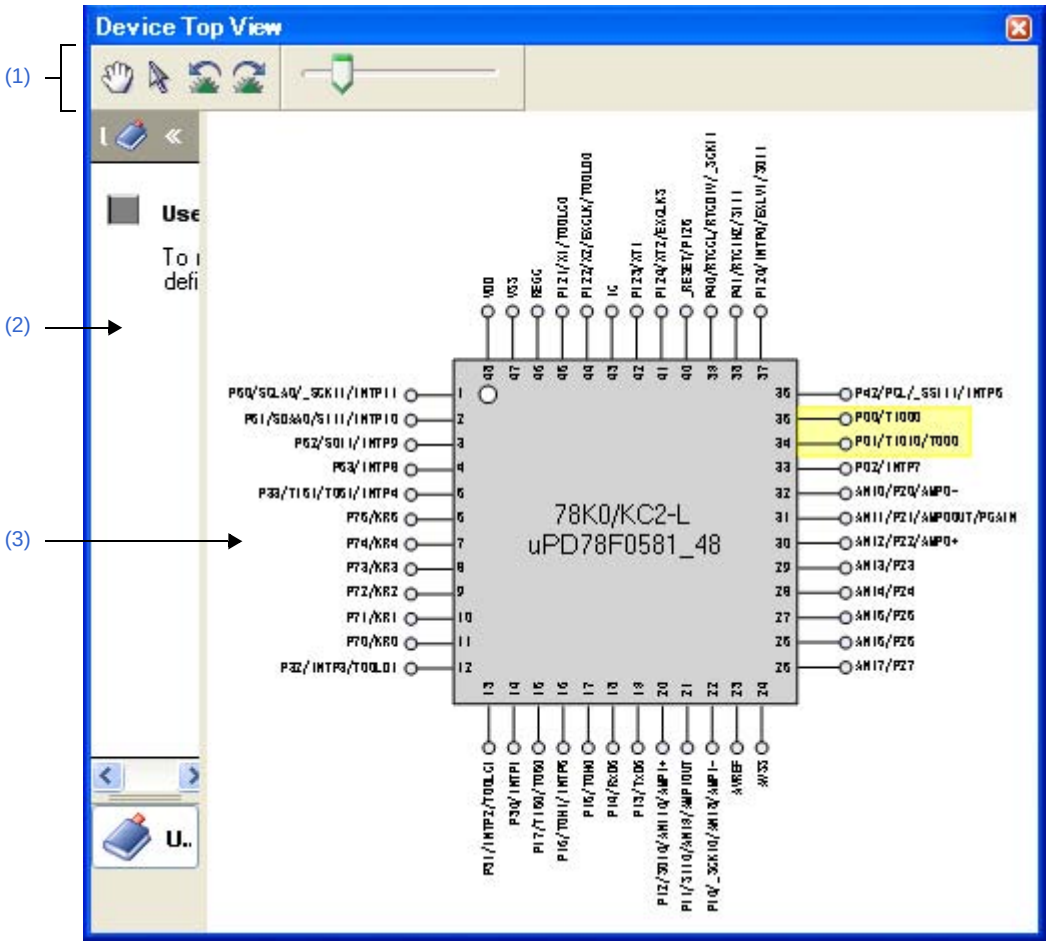
4. To add the "user's own column", use the New Column dialog box which opens by pressing the [New Column...] button in the Column Chooser dialog box which opens by pressing the  button in the upper left corner of the device pin list.

Device Top View panel

This panel displays the information entered in the Device Pin List panel.

Remark The Device top view area can be zoomed in and out by 100% in the tool bar.

Figure A-15. Device Top View Panel



The following items are explained here.

- [How to open]
- [Description of each area]
- [[File] menu (Device Top View panel-dedicated items)]
- [[Help] menu (Device Top View panel-dedicated items)]
- [Context menu]

[How to open]

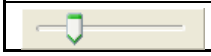
- On the Project Tree panel, double-click [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Top View].
- On the Project Tree panel, select [*Project name* (Project)] >> [Pin Configurator (Design Tool)] >> [Device Top View] and then press the [Enter] key.
- From the [View] menu, select [Pin Configurator] >> [Device Top View].

Remark In the Property panel, on the [Pin Configurator Settings] tab, if "BGA" is selected for the Package type, then this panel cannot be opened.

[Description of each area]

(1) Toolbar

This area consists of the following buttons.

	Clicks this button to enable changing of the display in the Device top view area by drag and drop. By pressing this button, the shape of the mouse cursor in the Device top view area changes from the arrow to the hand.
	Clicks this button to enable moving external peripheral components in the Device top view area to arbitrary locations, and select pins. By pressing this button, the shape of the mouse cursor which has changed into the hand by pressing the  button reverts back to the arrow.
	Rotates the content in the Device top view area 90 degrees counter-clockwise.
	Rotates the content in the Device top view area 90 degrees clockwise.
	Expands or reduces the content in the Device top view area.

(2) [User Define] area

Drag and drop the  button from this area to the Device top view area to creat and display an external peripheral controller.

(3) Device top view area

This area displays the pin assignment of the microcontroller.
Settings of the pin assignment are displayed using the colors specified by selecting [Device Top View Settings] tab >> [Color] in the Property panel.

Remark If the pin name in the diagram is double-clicked, the Device Pin List panel opens and the focus moves to the clicked pin in the list.

[[File] menu (Device Top View panel-dedicated items)]

Save Top View	Saves a report file (a file containing information configured using Pin Configurator: device top view) overwriting the existing file.
Save Top View As...	Opens the Save As dialog box for naming and saving a report file (a file containing information configured using Pin Configurator: device top view).

[[Help] menu (Device Top View panel-dedicated items)]

Open Help for Device Top View Panel	Displays the help of this panel.

[Context menu]

When you right click on a pin or external peripheral controller in the Device top view area, the following context menu displays.

(1) When a pin is right clicked

Use as	If the pin has multiple functions, select which function to use.
Connect to External Peripheral	Selects which external peripheral controller to connect the pin to.

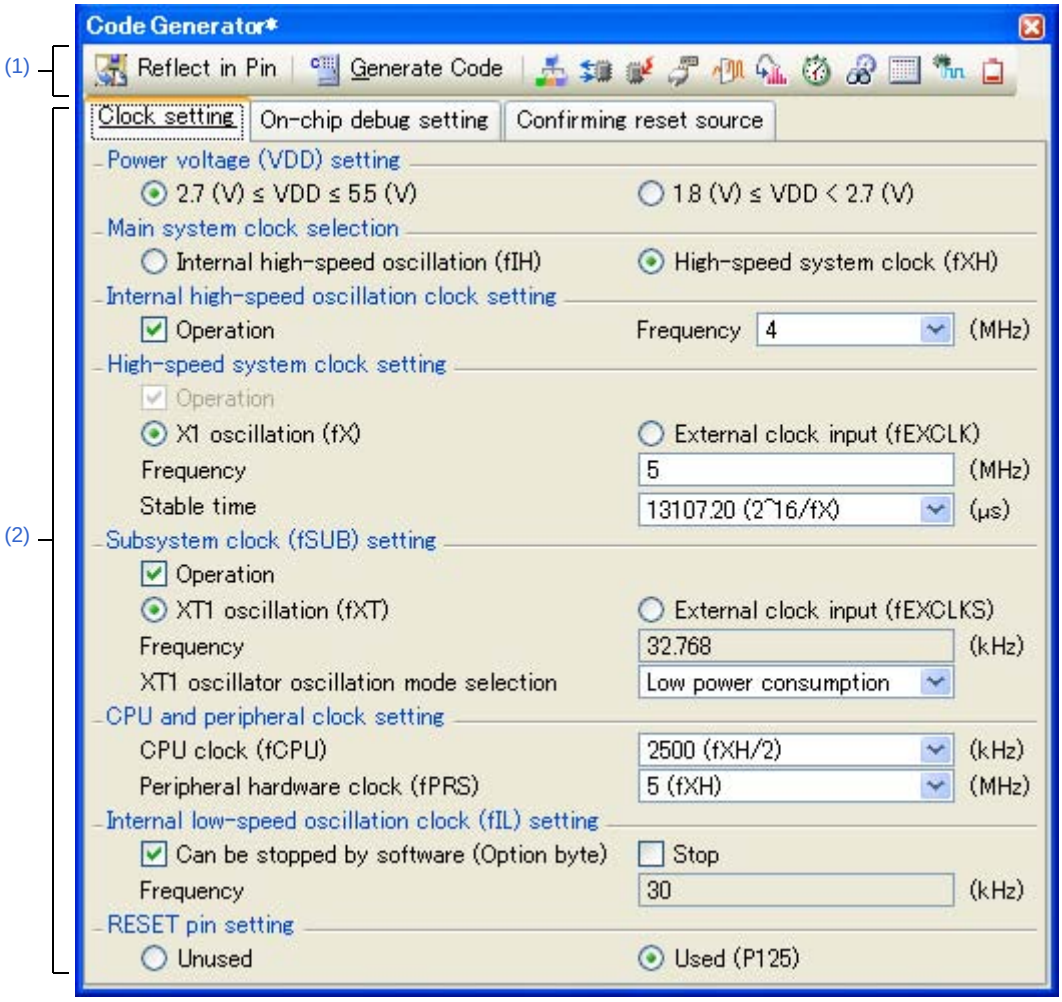
(2) When an external peripheral controller is right clocked

Disconnect Pin	Disconnects from the pin.
Delete External Peripheral	Removes the external peripheral controller.

Code Generator panel

This panel allows you to configure the information necessary to control the peripheral functions provided by the micro-controller.

Figure A-16. Code Generator Panel: [System]



The following items are explained here.

- [How to open]
- [Description of each area]
- [[File] menu (Code Generator panel-dedicated items)]
- [[Help] menu (Code Generator panel-dedicated items)]

[How to open]

- On the Project Tree panel, double-click [*Project name* (Project)] >> [Code Generator (Design Tool)] >> Peripheral function node "[System], [Port], etc.".
- On the Project Tree panel, select [*Project name* (Project)] >> [Code Generator (Design Tool)] >> Peripheral function node "[System], [Port], etc.", and then press the [Enter] key.

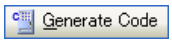
Remark If this panel is already open, pressing a different peripheral function button " , , etc." changes the content displayed in the Information setting area accordingly.

[Description of each area]

(1) Toolbar

This area consists of the following "peripheral function buttons".

When there is peripheral function target microcontroller is not supporting, peripheral functionbutton is not disokayed.

	Reflects settings made on this panel in the Device Pin List panel, and then output the changed contents to the Output panel. This button will be grayed out (disabled) if "Not reflected" is selected in the [PinPart Combination Mode] category of the [Generation] tab.
	Outputs the source code (device driver program) to the folder specified by selecting [Generation] tab >> [Output folder] in the Property panel.
	Changes the content displayed in the Information setting area to the "[System] for configuring the information necessary to control the functions of clock generator, on-chip debug function and etc. provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[Port] for configuring the information necessary to control the port functions provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[Interrupt] for configuring the information necessary to control the interrupt functions and the key interrupt function provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[Serial] for configuring the information necessary to control the functions of serial array unit and functions of serial interface provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[Operational Amplifier] for configuring the information necessary to control the functions of comparator/programmable gain amplifier provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[Comparator] for configuring the information necessary to control the function of comparator provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[A/D Converter] for configuring the information necessary to control the function of A/D converter provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[Timer] for configuring the information necessary to control the functions of timer array unit provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[Watchdog Timer] for configuring the information necessary to control the functions of watchdog timer provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[Real-time Clock] for configuring the information necessary to control the functions of real-time counter provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[Clock Output] for configuring the information necessary to control the functions of clock output controller provided by the microcontroller".
	Changes the content displayed in the Information setting area to the "[LVI] for configuring the information necessary to control the functions of low-voltage detector provided by the microcontroller".

(2) Information setting area

The content displayed in this area differs depending on the "peripheral function node" or "peripheral function button" selected or pressed when opening this panel.

See User's Manual for Microcontroller for details on the items to be set.

[[File] menu (Code Generator panel-dedicated items)]

Save Code Generator Report	Outputs report files (a file containing information configured using Code Generator and a file containing information regarding the source code).

Remarks 1. The output format for the report file (either HTML or CSV) is selected by clicking [Generation] tab >> [Report type] in the Property panel.

2. The destination folder for the report file is specified by clicking [Generation] tab >> [Output folder] in the Property panel.

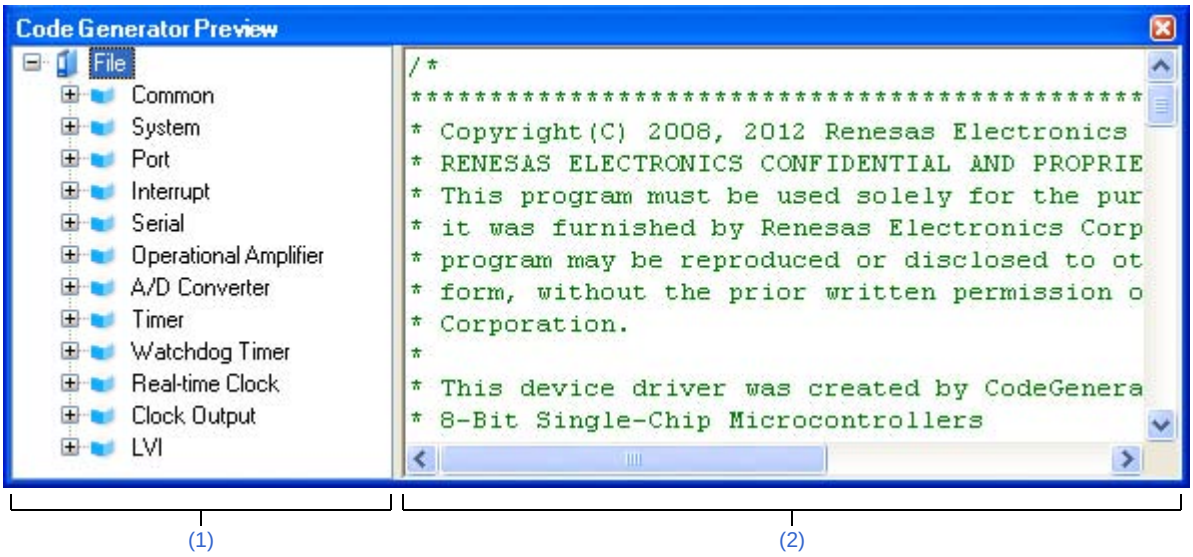
[[Help] menu (Code Generator panel-dedicated items)]

Open Help for Code Generator Panel	Displays the help of this panel.

Code Generator Preview panel

This panel allows you to confirm or configure on a per-API function basis the setting that determines whether or not the source code (device driver program) is generated when the Generate Code button is pressed in the Code Generator panel. It also allows you to confirm the source code that reflects the information configured in the Code Generator panel.

Figure A-17. Code Generator Preview Panel



The following items are explained here.

- [How to open]
- [Description of each area]
- [[File] menu (Code Generator Preview panel-dedicated items)]
- [[Help] menu (Code Generator Preview panel-dedicated items)]
- [Context menu]

[How to open]

- From the [View] menu, select [Code Generator Preview].

[Description of each area]

(1) Preview tree

This area allows you to confirm or configure on a per-API function basis the setting that determines whether or not the source code (device driver program) is generated when the Generate Code button is pressed in the Code Generator panel.

Remarks 1. You can change the source code to be displayed by selecting the source file name or API function name in this tree.

2. To select whether or not to generate the source code, use the context menu (Generate code/Not generate code) which is displayed by right-clicking the mouse while the mouse cursor is on the desired icon in the tree.

3. You can confirm the current setting that determines whether or not to generate the source code by checking the type of icon.

Table A-2. Setting That Determines Whether or Not to Generate the Source Code

Type of Icon	Outline
	Source code for the currently selected API function is generated. If this icon is displayed next to the API function, the corresponding source code must be generated (it is impossible to change the icon to ).
	Source code for the currently selected API function is generated.
	Source code for the currently selected API function is not generated.

(2) Source code display area

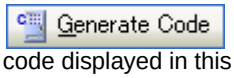
This area allows you to confirm the source code (device driver program) that reflects the information configured in the Code Generator panel.

The following table displays the meaning of the color of the source code text displayed in this area.

Table A-3. Color of Source Code

Color	Outline
Green	Comment
Blue	Reserved word for C compiler
Red	Numeric value
Black	Code section
Gray	File name

Remarks 1. You cannot edit the source code within this panel.

2. For some of the API functions (such as API functions for serial array units), values such as the SFR register value are calculated and finalized when the source code is generated (when the  button on the Code Generator panel is pressed). For this reason, the source code displayed in this panel may not be the same as that would actually be generated.
3. You can change the source code to be displayed by selecting the source file name or API function name in the preview tree.

[[File] menu (Code Generator Preview panel-dedicated items)]

Save Code Generator Report	Outputs report files (a file containing information configured using Code Generator and a file containing information regarding the source code).

Remarks 1. The output format for the report file (either HTML or CSV) is selected by clicking [Generation] tab >> [Report type] in the Property panel.

2. The destination folder for the report file is specified by clicking [Generation] tab >> [Output folder] in the Property panel.

[[Help] menu (Code Generator Preview panel-dedicated items)]

Open Help for Code Generator Preview Panel	Displays the help of this panel.

[Context menu]

The following context menu items are displayed by right clicking the mouse.

Generate code	Makes a setting so that the source code of the currently selected API function is generated to the folder specified by selecting [Generation] tab >> [Output folder] in the Property panel. Selecting this context menu item changes the icon of the currently selected API function from  to  . This item will be grayed out (disabled) if the currently selected API function is not initialization API function, and "Output only initialization API function" is selected [Generation] tab >> [Output control of API function] in the Property panel.
Not generate code	Makes a setting so that the source code of the currently selected API function is not generated when the  button is pressed in the Code Generator panel. Selecting this context menu item changes the icon of the currently selected API function from  to  .
Rename	Selecting this menu item changes the name portion of the currently selected file or API function into an edit box for editing the name. You can change the name of the file or API function by editing its name in the edit box.
Default	Reverts the file name or API function name to its original name before it was edited.
Property	Opens the Property panel that contains the information for the currently selected file.

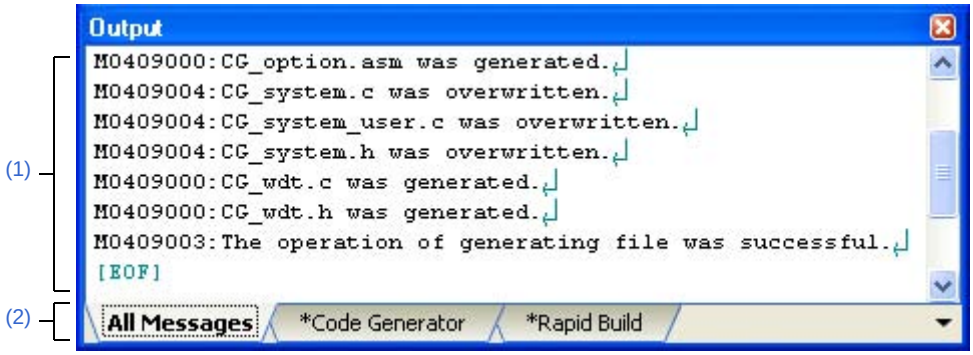
Output panel

This panel is used to display operating logs for various components (design tool, build tool, debug tool, etc.) provided by CubeSuite+.

The messages are dassified by the message origination tool and displated on the individual tabs.

Remark The Message area can be zoomed in and out by 100% in the tool bar, or by operating the mouse wheel while holding down the [Ctrl] key.

Figure A-18. Output Panel



The following items are explained here.

- [How to open]
- [Description of each area]
- [[File] menu (Output panel-dedicated items)]
- [[Edit] menu (Output panel-dedicated items)]
- [Context menu]

[How to open]

- From the [View] menu, select [Output].

[Description of each area]

(1) Message area

The output messages of each tool are displayed.

The colors of message display differ with the type of message as shown below (character colors and background colors depend on the configuration in the [General - Font and Color] category of the Option dialog box).

Message Type	Display Example (Default)			Description
Normal message	ABCD abcd 0123	Character color	Black	Displayed with information notices.
		Background color	White	
Warning message	ABCD abcd 0123	Character color	Bule	Displayed with warnings about operations.
		Background color	Standard color	
Error message	ABCD abcd 0123	Character color	Red	Displayed when there is a critical error, or when execution is not possible due to a operational mistake.
		Background color	Light gray	

(2) Tab selection area

Select the tab that indicates the origin of message.

The following tabs are available for the debug tool.

Tab Name	Description
All Messages	Displays operation logs for all components (design tool, build tool, debug tool, etc.) provided by CubeSuite+ in order of output.
Code Generator	Display only operation logs for the Code Generator out of those for various components (design tool, build tool, debug tool, etc.) provided by CubeSuite+.

Caution Even if a new message is output on a deselected tab, tab selection will not automatically switch. In this case, " * " mark will be added in front of the tab name, indicating that a new message has been output.

[[File] menu (Output panel-dedicated items)]

Save Output-*Tab Name*	Saves the message corresponding to the specified tab overwriting the existing file.
Save Output-*Tab Name* As...	Opens the Save As dialog box for naming and saving the message corresponding to the specified tab.

[[Edit] menu (Output panel-dedicated items)]

Copy	Sends the character string or lines selected with range selection to the clipboard.
Select All	Selects all the messages displayed on the Message area.
Search...	Opens the Search and Replace dialog box for searching strings with the [Quick Search] tab selected.
Replace...	Opens the Search and Replace dialog box for replacing strings with the [Whole Replace] tab selected.

[Context menu]

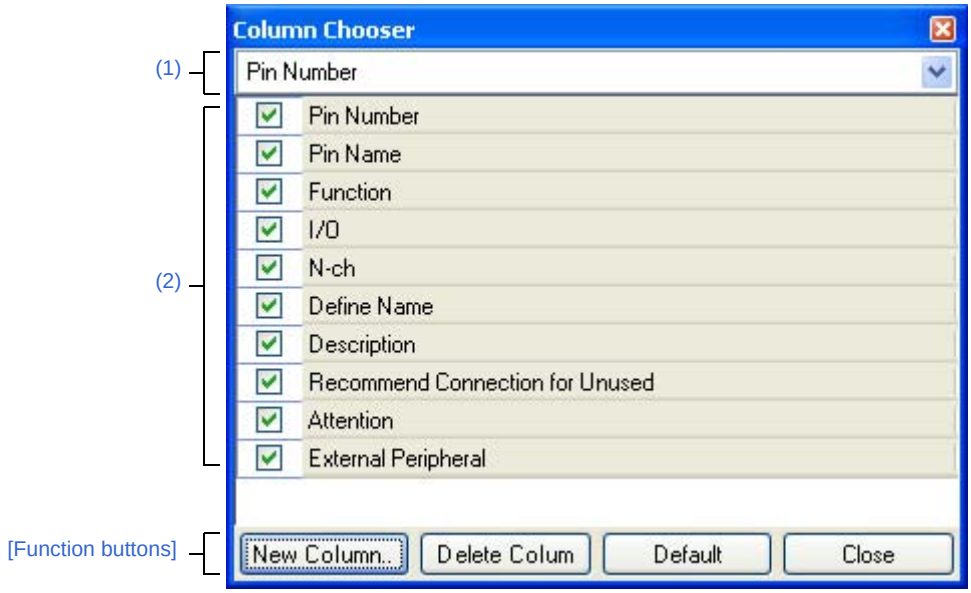
The following context menu items are displayed by right clicking the mouse.

Copy	Sends the character string or lines selected with range selection to the clipboard.
Select All	Selects all the messages displayed on the Message area.
Clear	Deletes all the messages displayed on the Message area.
Stop Searching	Cancels the search currently being executed. This is invalid when a search is not being executed.
Open Help for Message	Displays help for the message on the current caret location. This only applies to warning messages and error messages.

Column Chooser dialog box

This dialog box allows you to choose whether or not to display the item listed in this dialog box in the device pin list, and add columns to or delete columns from the device pin list.

Figure A-19. Column Chooser Dialog Box



The following items are explained here.

- [How to open]
- [Description of each area]
- [Function buttons]

[How to open]

- In the [Pin Number] tab of the Device Pin List panel, click the button.
- In the [Macro] tab of the Device Pin List panel, click the button.
- In the [External Peripheral] tab of the Device Pin List panel, click the button.

[Description of each area]

(1) Operational object selection area

This area allows you to select the device pin list to be configured in this dialog box.

Pin Number	Configures the device pin list corresponding to the [Pin Number] tab.
Macro	Configures the device pin list belonging to the first layer of the [Macro] tab.
Macro - Pin	Configures the device pin list belonging to the second layer of the [Macro] tab.
External Peripheral	Configures the device pin list belonging to the first layer of the [External Peripheral] tab.
External Peripheral - Pin	Configures the device pin list belonging to the second layer of the [External Peripheral] tab.

Figure A-20. Operational Object ([Pin Number] Tab)

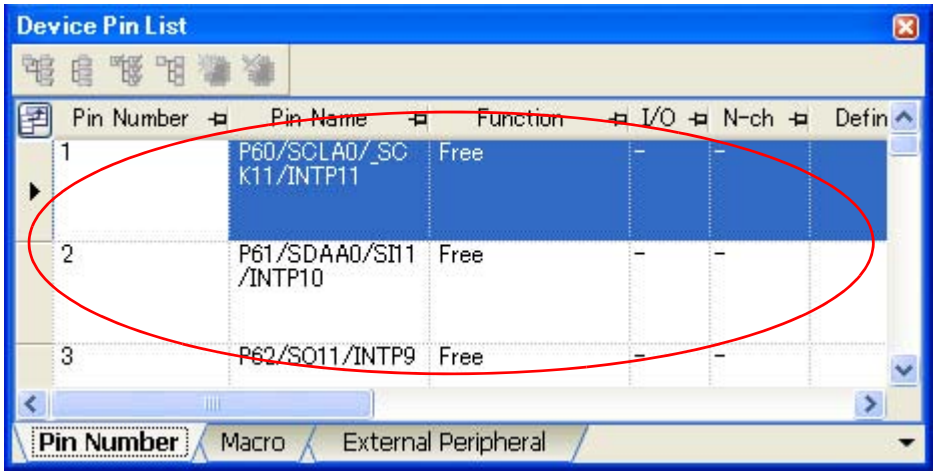


Figure A-21. Operational Object ([Macro] Tab: First Layer)

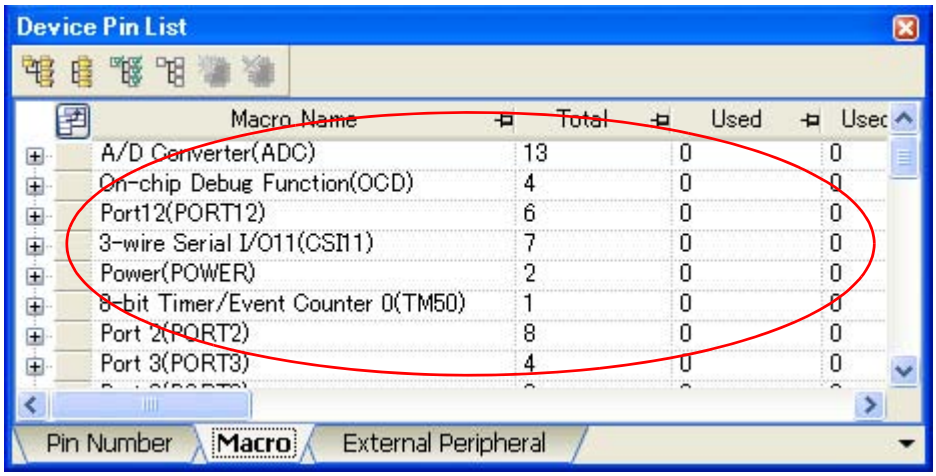


Figure A-22. Operational Object ([Macro] Tab: Second Layer)

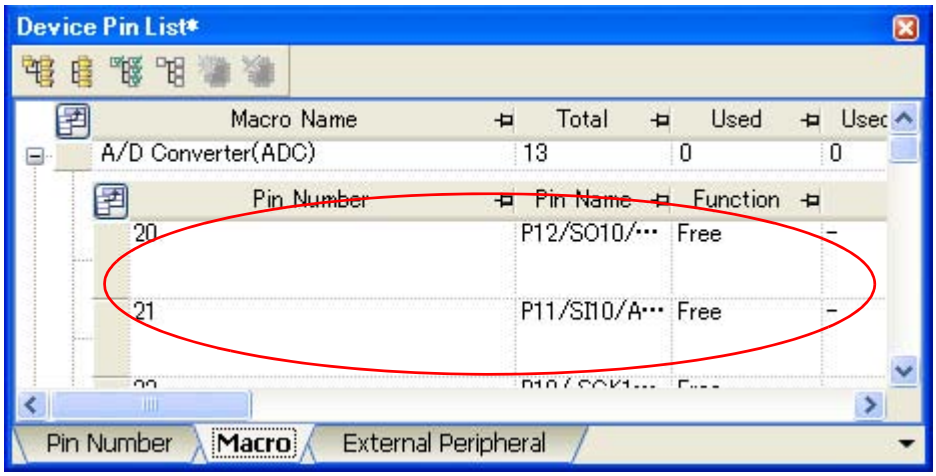


Figure A-23. Operational Object ([External Peripheral] Tab: First Layer)

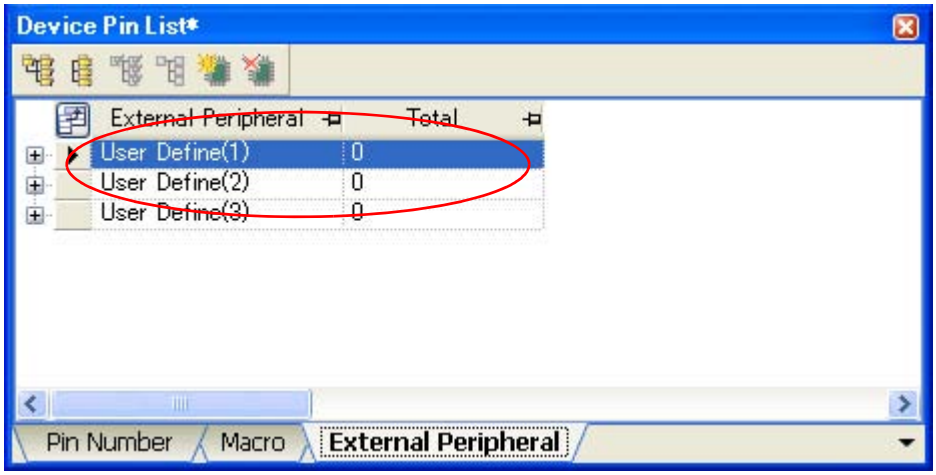
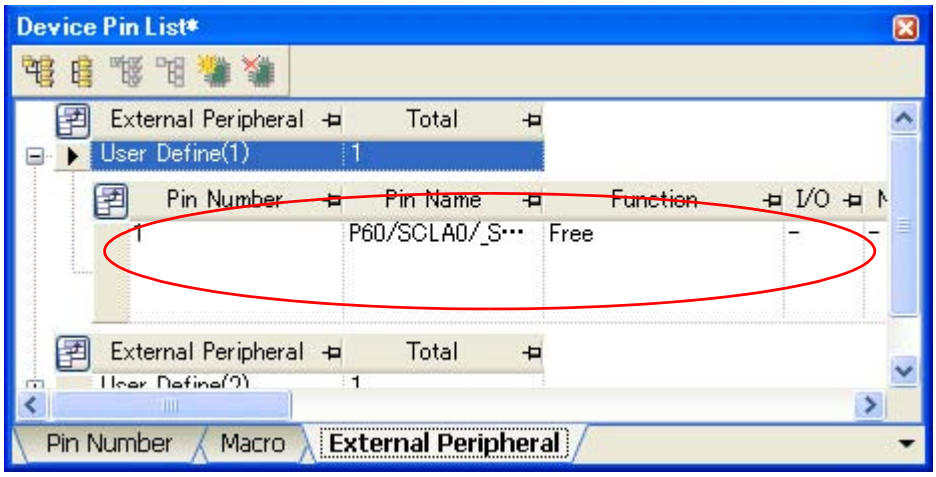


Figure A-24. Operational Object ([External Peripheral] Tab: Second Layer)



(2) Displayed item selection area

Select whether or not to display the item selected in the Operational object selection area in the device pin list.

Checked	Displays the selected item in the device pin list.
Not checked	Hides the selected item in the device pin list.

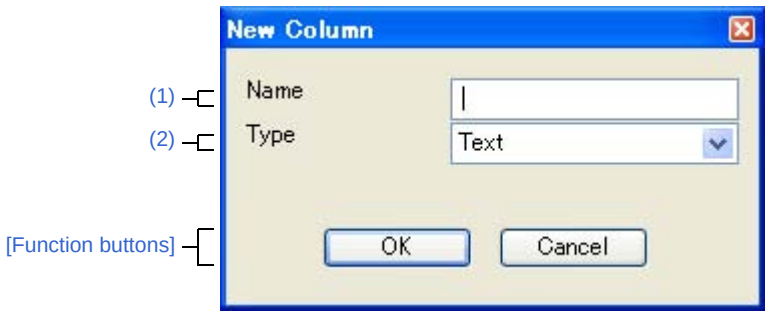
[Function buttons]

Button	Function
New Column...	Opens the New Column dialog box for adding columns to the device pin list.
Delete Column	Deletes the selected columns from the device pin list. You can only delete the column which you added using the New Column dialog box.
Default	Restores the column order to the default settings.
Close	Closes this dialog box.

New Column dialog box

This dialog box allows you to add your own column to the device pin list.

Figure A-25. New Column Dialog Box



The following items are explained here.

- [How to open]
- [Description of each area]
- [Function buttons]

[How to open]

- Click the [New Column...] button in the Column Chooser dialog box.

[Description of each area]

(1) [Name]

This area allows you to enter column headings of the columns added to the device pin list.
Within 256 characters can be entered in the [Name].

(2) [Type]

Select the input format of the column to add to the device pin list.

Text	Only character strings can be entered in the column.
Cehck box	Adds a column of check boxes.
Whole number	Only integers can be entered in the column.
Real number	Only real numbers can be entered in the column.
Date	Only dates in YYYYMMDD format can be entered in the column.

[Function buttons]

Button	Function
OK	Adds a column that has the column heading specified in the [Name] to the right end of the device pin list.
Cancel	Ignores the setting and closes this dialog box.

Browse For Folder dialog box

This dialog box allows you to specify the output destination for files (source code, report file, etc.).

Figure A-26. Browse For Folder Dialog Box

The following items are explained here.

- [How to open]
- [Description of each area]
- [Function buttons]

[How to open]

- In the [Generation] tab of the Property panel, click the [...] button in [Output folder].

[Description of each area]

(1) Folder location

Select the folder to which the files (source code, report file, etc.) are output.

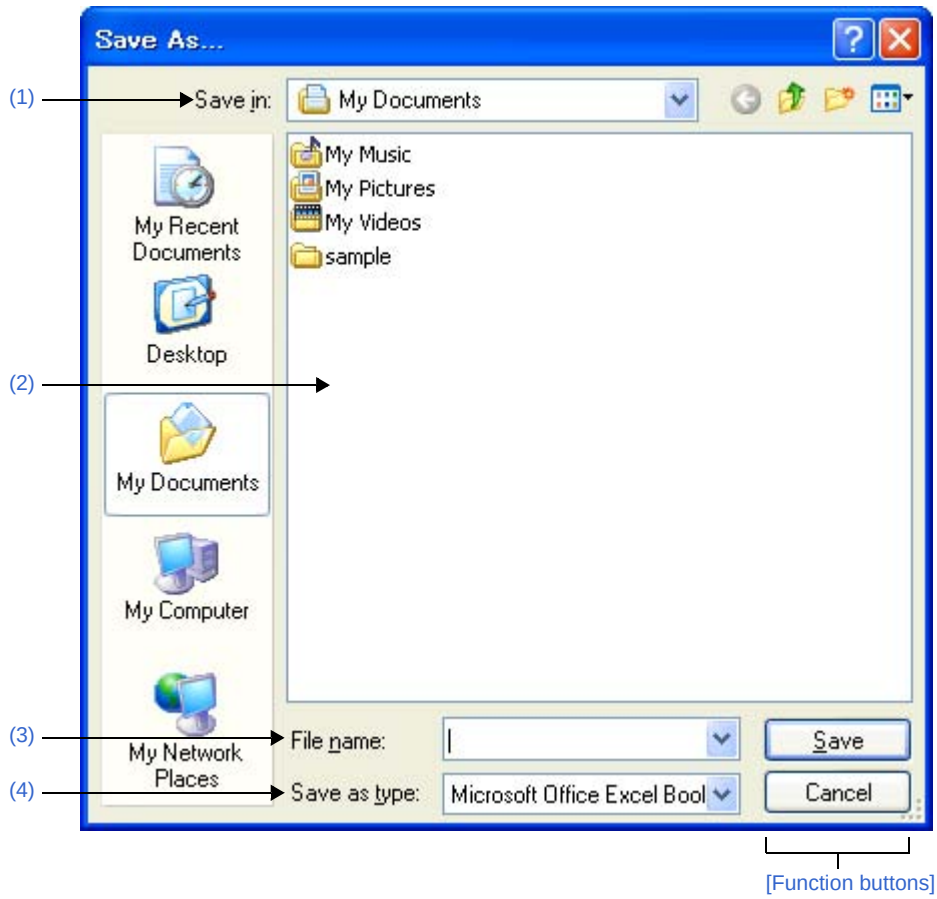
[Function buttons]

Button	Function
Make New Folder	Creates a "New Folder" below the folder selected in the Folder location.
OK	Specifies the folder selected in the Folder location as the destination for the files.
Cancel	Ignores the setting and closes this dialog box.

Save As dialog box

This dialog box allows you to name and save a file (such as a report file).

Figure A-27. Save As Dialog Box



The following items are explained here.

- [How to open]
- [Description of each area]
- [Function buttons]

[How to open]

- From the [File] menu, select [Save <*object*> As...].

[Description of each area]

(1) [Save in]

Select the folder to which the files (report files, etc.) are output.

(2) List of files

This area displays a list of files matching the conditions selected in [Save in] and [Save as type].

(3) [File name]

Specify the name of the file to be output.

(4) [Save as type]

Select the type of the file to be output.

Microsoft Office Excel Book (*.xls)	Microsoft Office Excel Book format
Bitmap (*.bmp)	Bitmap format
PNG (*.png)	PNG format
JPEG (*.jpg)	JPEG format
EMF (*.emf)	EMF format

[Function buttons]

Button	Function
Save	Outputs a file having the name specified in the [File name] and [Save as type] to the folder specified in the [Save in].
Cancel	Ignores the setting and closes this dialog box.

APPENDIX B OUTPUT FILES

This appendix describes the files output by Code Generator.

B.1 Overview

Below is a list of files output by Code Generator.

Table B-1. File List

Unit of Output	File Name	Description
Peripheral function	CG_*PeripheralFunctionName*.c	Initial function, API function
	CG_*PeripheralFunctionName*_user.c	Interrupt function (MD_INT*xxx*), callback function
	CG_*PeripheralFunctionName*.h	Defines macros for assigning values to registers
Project	CG_main.c	main function, R_MAIN_UserInit function
	CG_systeminit.c	Call initial function of peripheral function Call CG_ReadResetSource
	CG_macrodriver.h	Defines common macros used by all source files
	CG_userdefine.h	Empty file (for user definitions)
	CG_lk.dr	Link directive
	CG_option.asm	Option bytes, secures ROM for MINICUBE2

B.2 Output File

Below are the files (peripheral function) output by Code Generator.

Table B-2. File List (Peripheral Function)

Peripheral Function	Source File Name	Names of API Functions Included
System	CG_system.c	CLOCK_Init CG_ChangeClockMode CG_ChangeFrequency CG_SelectPowerSaveMode CG_SelectStabTime CG_ChangePllMode
	CG_system_user.c	CLOCK_UserInit CG_ReadResetSource
	CG_system.h	-
Port	CG_port.c	PORT_Init PORT_ChangePmnInput PORT_ChangePmnOutput
	CG_port_user.c	PORT_UserInit
	CG_port.h	-
Interrupt	CG_int.c	INTP_Init KEY_Init INT_MaskableInterruptEnable INTPn_Disable

Peripheral Function	Source File Name	Names of API Functions Included
Interrupt	CG_int.c	INTPn_Enable KEY_Disable KEY_Enable
	CG_int_user.c	INTP_UserInit KEY_UserInit MD_INTP*n* MD_INTKR
	CG_int.h	-
Serial	CG_serial.c	UART6_Init UART6_Start UART6_Stop UART6_SendData UART6_ReceiveData CSI1n_Init CSI1n_Start CSI1n_Stop CSI1n_ReceiveData CSI1n_SendReceiveData IICA_Init IICA_Stop IICA_MasterSendStart IICA_MasterReceiveStart IICA_StopCondition IICA_SlaveSendStart IICA_SlaveReceiveStart
	CG_serial_user.c	UART6_UserInit UART6_SendEndCallback UART6_ReceiveEndCallback UART6_SoftOverRunCallback UART6_ErrorCallback CSI1n_UserInit CSI1n_SendEndCallback CSI1n_ReceiveEndCallback IICA_UserInit IICA_MasterSendEndCallback IICA_MasterReceiveEndCallback IICA_MasterErrorCallback IICA_SlaveSendEndCallback IICA_SlaveReceiveEndCallback IICA_SlaveErrorCallback IICA_GetStopConditionCallback MD_INTSR6 MD_INTSRE6 MD_INTST6 MD_INTCSI1*n* MD_INTIICA0

Peripheral Function	Source File Name	Names of API Functions Included
Serial	CG_serial.h	-
Operational Amplifier	CG_opamp.c	OPAMP_Init PGA_Start PGA_Stop PGA_ChangePGAFactor AMP_Start AMP_Stop AMPn_Start AMPn_Stop
	CG_opamp_user.c	OPAMP_UserInit
	CG_opamp.h	-
Comparator	CG_comparator.c	Comparator_Init Comparatorn_Start Comparatorn_Stop
	CG_comparator_user.c	Comparator_UserInit MD_INTCMP*n*
	CG_comparator.h	-
A/D Converter	CG_ad.c	AD_Init 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	-
Timer	CG_timer.c	TMX_Init TMXn_Start TMXn_Stop TMXn_ChangeDuty TMXn_ChangeDualDuty TMX_EnableHighImpedanceState TMX_DisableHighImpedanceState TM00_Init TM00_Start TM00_Stop TM00_ChangeTimerCondition TM00_GetFreeRunningValue TM00_SoftwareTriggerOn TM00_ChangeDuty TM00_GetPulseWidth TM5n_Init TM5n_Start

Peripheral Function	Source File Name	Names of API Functions Included
Timer	CG_timer.c	TM5n_Stop TM5n_ChangeTimerCondition TM5n_ChangeDuty TMHn_Init TMHn_Start TMHn_Stop TMHn_ChangeTimerCondition TMHn_ChangeDuty TMH1_CarrierOutputEnable TMH1_CarrierOutputDisable
	CG_timer_user.c	TM00_UserInit TM5n_UserInit TMHn_UserInit MD_INTTMX*n* MD_INTTM0*n*0 MD_INTTM5*n* MD_INTMH*n*
	CG_timer.h	-
Watchdog Timer	CG_wdt.c	WDT_Restart
	CG_wdt.h	-
Real-time Clock	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 RTC_RTC1HZ_OutputDisable RTC_RTCCL_OutputEnable RTC_RTCCL_OutputDisable RTC_RTCDIV_OutputEnable RTC_RTCDIV_OutputDisable RTC_ChangeCorrectionValue
	CG_rtc_user.c	RTC_UserInit RTC_ConstPeriodInterruptCallback

Peripheral Function	Source File Name	Names of API Functions Included
Real-time Clock	CG_rtc_user.c	RTC_AlarmInterruptCallback MD_INTRTC MD_INTRTCl
	CG_rtc.h	-
Clock Output	CG_pcl.c	PCL_Init PCL_Start PCL_Stop PCL_ChangeFreq
	CG_pcl_user.c	PCL_UserInit
	CG_pcl.h	-
LVI	CG_lvi.c	LVI_Init LVI_InterruptModeStart LVI_ResetModeStart LVI_Stop LVI_SetLVILevel
	CG_lvi_user.c	LVI_UserInit MD_INTLVI
	CG_lvi.h	-

APPENDIX C API FUNCTIONS

This appendix describes the API functions output by Code Generator.

C.1 Overview

Below are the naming conventions for API functions output by Code Generator.

- Macro names are in ALL CAPS.
  The number in front of the macro name is a hexadecimal value; this is the same value as the macro value.
- Local variable names are in all lower case.
- Global variable names start with a "g" and use Camel Case.
- Names of pointers to global variables start with a "gp" and use Camel Case.
- Names of elements in enum statements are in ALL CAPS.

C.2 Output Function

Below is a list of API functions output by Code Generator.

Table C-1. API Function List

Peripheral Function	API Function Name	Function
System	CLOCK_Init	Performs initialization required to control the clock generator, on-chip debug, and etc. .
	CLOCK_UserInit	Performs user-defined initialization relating to the clock generator, on-chip debug, and etc. .
	CG_ReadResetSource	Performs processing in response to RESET signal.
	CG_ChangeClockMode	Changes the CPU clock/peripheral hardware clock.
	CG_ChangeFrequency	Changes the division ratio of the CPU clock/peripheral hardware clock.
	CG_SelectPowerSaveMode	Configures the CPU's standby function.
	CG_SelectStabTime	Configures the oscillation stabilization time of the X1 clock.
	CG_ChangePllMode	Controls the operation of PLL function.
Port	PORT_Init	Performs initialization necessary to control port functions.
	PORT_UserInit	Performs user-defined initialization relating to the port.
	PORT_ChangePmnInput	Switches the pin's I/O mode from output mode to input mode.
	PORT_ChangePmnOutput	Switches the pin's I/O mode from input mode to output mode.
Interrupt	INTP_Init	Performs initialization necessary to control the external interrupt INTP*n* functions.
	INTP_UserInit	Performs user-defined initialization relating to the external interrupt INTP*n* functions.
	KEY_Init	Performs initialization necessary to control the key interrupt INTKR functions.
	KEY_UserInit	Performs user-defined initialization relating to the key interrupt INTKR functions.
	INT_MaskableInterruptEnable	Disables/enables the acceptance of the maskable interrupts.
	INTPn_Disable	Disables the acceptance of the maskable interrupts INTP*n* (external interrupt requests).

Peripheral Function	API Function Name	Function
Interrupt	INTPn_Enable	Enables the acceptance of the maskable interrupts INTP*n* (external interrupt requests).
	KEY_Disable	Disables the acceptance of the key interrupts INTKR.
	KEY_Enable	Enables the acceptance of the key interrupts INTKR.
Serial	UART6_Init	Performs initialization of the serial interface (UART6) channel.
	UART6_UserInit	Performs user-defined initialization of the serial interface (UART6).
	UART6_Start	Sets UART communication to standby mode.
	UART6_Stop	Ends UART communication.
	UART6_SendData	Starts UART data transmission.
	UART6_ReceiveData	Starts UART data reception.
	UART6_SendEndCallback	Performs processing in response to the UART transmission complete interrupt INTST6.
	UART6_ReceiveEndCallback	Performs processing in response to the UART reception complete interrupt INTSR6.
	UART6_SoftOverRunCallback	Performs processing in response to the UART reception complete interrupt INTSR6.
	UART6_ErrorCallback	Performs processing in response to the UART communication error interrupt INTSRE6.
	CSI1n_Init	Performs initialization of the serial interface (CSI1*n*) channel.
	CSI1n_UserInit	Performs user-defined initialization of the serial interface (CSI1*n*).
	CSI1n_Start	Sets CSI1*n* communication to standby mode.
	CSI1n_Stop	Ends CSI1*n* communication.
	CSI1n_ReceiveData	Starts CSI1*n* data reception.
	CSI1n_SendReceiveData	Starts CSI1*n* data transmission/reception.
	CSI1n_SendEndCallback	Performs processing in response to the CSI1*n* communication complete interrupt INTCSI1*n*.
	CSI1n_ReceiveEndCallback	Performs processing in response to the CSI1*n* communication complete interrupt INTCSI1*n*.
	IICA_Init	Performs initialization of the serial interface (IICA).
	IICA_UserInit	Performs user-defined initialization of the serial interface (IICA).
	IICA_Stop	Ends IICA communication.
	IICA_MasterSendStart	Starts IICA master transmission.
	IICA_MasterReceiveStart	Starts IICA master reception.
	IICA_StopCondition	Generates stop conditions.
	IICA_MasterSendEndCallback	Performs processing in response to the IICA communication complete interrupt INTIICA0.
	IICA_MasterReceiveEndCallback	Performs processing in response to the IICA communication complete interrupt INTIICA0.

Peripheral Function	API Function Name	Function
Serial	IICA_MasterErrorCallback	Performs processing in response to detection of error in IICA master communication.
	IICA_SlaveSendStart	Starts IICA slave transmission.
	IICA_SlaveReceiveStart	Starts IICA slave reception.
	IICA_SlaveSendEndCallback	Performs processing in response to the IICA communication complete interrupt INTIICA0.
	IICA_SlaveReceiveEndCallback	Performs processing in response to the IICA communication complete interrupt INTIICA0.
	IICA_SlaveErrorCallback	Performs processing in response to detection of error in IICA slave communication.
	IICA_GetStopConditionCallback	Performs processing in response to detection of stop condition in IICA slave communication.
Operational Amplifier	OPAMP_Init	Performs initialization necessary to control operational amplifier functions.
	OPAMP_UserInit	Performs user-defined initialization relating to the operational amplifier.
	PGA_Start	Starts the operation of operational amplifier (PGA mode).
	PGA_Stop	Ends the operation of operational amplifier (PGA mode).
	PGA_ChangePGAFactor	Sets the input voltage amplification factor of a operational amplifier (PGA mode).
	AMP_Start	Starts the operation of operational amplifier (single AMP mode).
	AMP_Stop	Ends the operation of operational amplifier (single AMP mode).
	AMPn_Start	Starts the operation of operational amplifier *n* (single AMP mode).
	AMPn_Stop	Ends the operation of operational amplifier *n* (single AMP mode).
Comparator	Comparator_Init	Performs initialization necessary to control comparator functions.
	Comparator_UserInit	Performs user-defined initialization relating to the comparator.
	Comparatorn_Start	Starts the operation of comparator *n*.
	Comparatorn_Stop	Ends the operation of comparator *n*.
A/D Converter	AD_Init	Performs initialization necessary to control A/D converter functions.
	AD_UserInit	Performs user-defined initialization relating to the A/D converter.
	AD_ComparatorOn	Enables operation of voltage converter.
	AD_ComparatorOff	Disables operation of voltage converter.
	AD_Start	Starts A/D conversion.
	AD_Stop	Ends A/D conversion.
	AD_SelectADChannel	Configures the analog voltage input pin for A/D conversion.

Peripheral Function	API Function Name	Function
A/D Converter	AD_Read	Reads the results of A/D conversion (10 bits).
	AD_ReadByte	Reads the results of A/D conversion (8 bits; most significant 8 bits of 10-bit resolution).
Timer	TMX_Init	Performs initialization necessary to control 16-bit timer X*n* functions.
	TMXn_Start	Starts the count for 16-bit timer X*n*.
	TMXn_Stop	Ends the count for 16-bit timer X*n*.
	TMXn_ChangeDuty	Changes the duty ratio of the PWM signal single-output to the TOX0*n* pin.
	TMXn_ChangeDualDuty	Changes the duty ratio of the PWM signal dulal-output to the TOX0*n* pin.
	TMX_EnableHighImpedanceState	Begins high impedance output of the 16-bit timer X*n*.
	TMX_DisableHighImpedanceState	Ends high impedance output of the 16-bit timer X*n*.
	TM00_Init	Performs initialization necessary to control 16-bit timer/event counter 00 functions.
	TM00_UserInit	Performs user-defined initialization relating to the 16-bit timer/ event counter 00.
	TM00_Start	Starts the count for 16-bit timer/event counter 00.
	TM00_Stop	Ends the count for 16-bit timer/event counter 00.
	TM00_ChangeTimerCondition	Changes the value of capture/compare control register 00 (CRC00).
	TM00_GetFreeRunningValue	Captures the content of the capture register (CR0*n*0).
	TM00_SoftwareTriggerOn	Generates the trigger (software trigger) for one-shot pulse output.
	TM00_ChangeDuty	Changes the duty ratio of the signal output to the TO00 pin.
	TM00_GetPulseWidth	Captures the high/low-level width measured for the signal (pulses) input to the TI0*n*0 pin.
	TM5n_Init	Performs initialization necessary to control 8-bit timer/event counter 5*n* functions.
	TM5n_UserInit	Performs user-defined initialization relating to the 8-bit timer/ event counter 5*n*.
	TM5n_Start	Starts the count for 8-bit timer/event counter 5*n*.
	TM5n_Stop	Ends the count for 8-bit timer/event counter 5*n*.
	TM5n_ChangeTimerCondition	Changes the value of 8-bit timer compare register 5*n* (CR5*n*).
	TM5n_ChangeDuty	Changes the duty ratio of the PWM signal output to the TO5*n* pin.
	TMHn_Init	Performs initialization necessary to control 8-bit timer H*n* functions.
	TMHn_UserInit	Performs user-defined initialization relating to the 8-bit timer H*n*.
	TMHn_Start	Starts the count for 8-bit timer H*n*.
	TMHn_Stop	Ends the count for 8-bit timer H*n*.

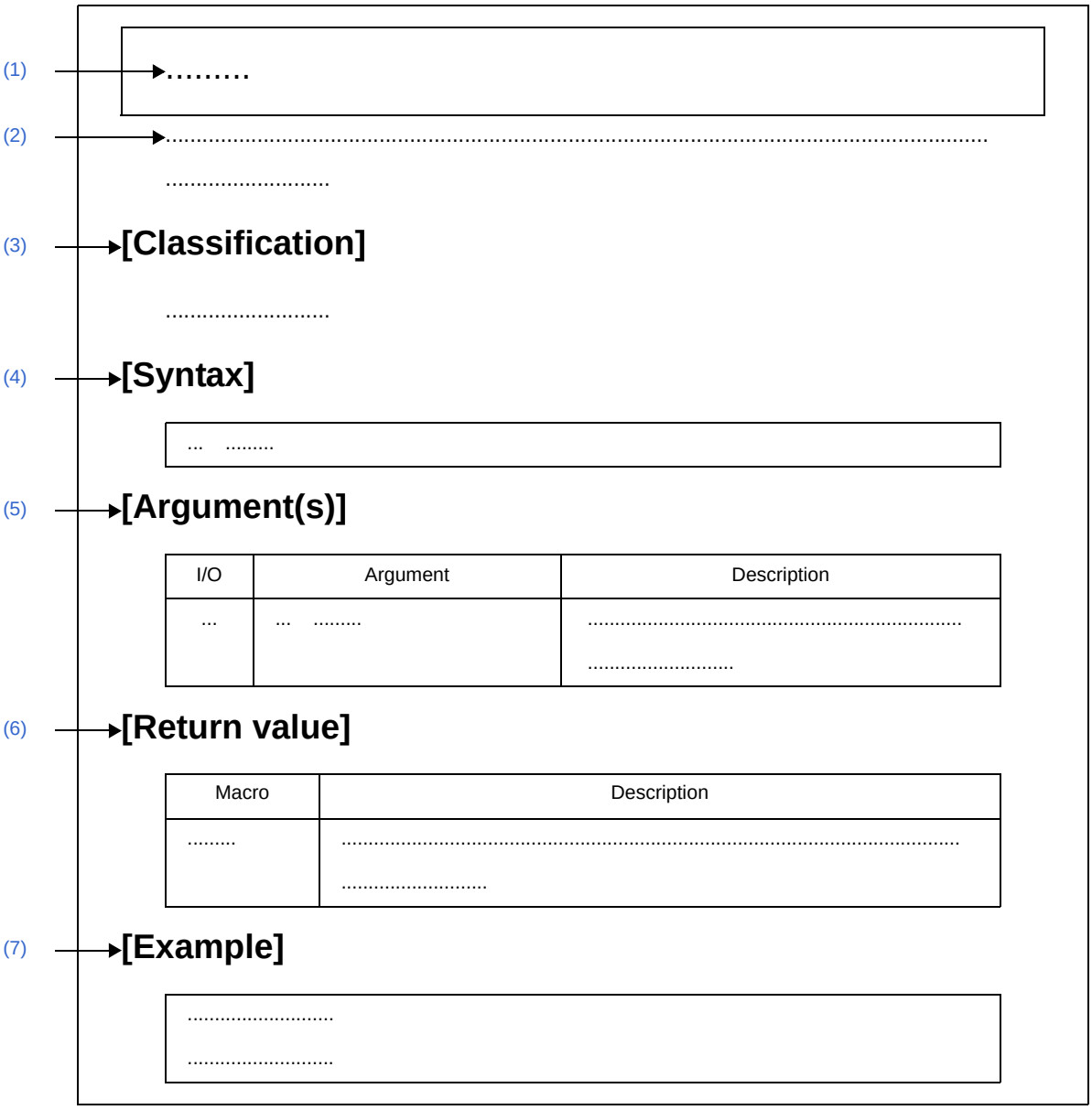
Peripheral Function	API Function Name	Function
Timer	TMHn_ChangeTimerCondition	Changes the value of 8-bit timer H compare register 0*n*/1*n* (CMP0*n*/CMP1*n*).
	TMHn_ChangeDuty	Changes the duty ratio of the PWM signal output to the TOH*n* pin.
	TMH1_CarrierOutputEnable	Begins carrier pulse output of the 8-bit timer H1 (carrier generator mode).
	TMH1_CarrierOutputDisable	Ends carrier pulse output of the 8-bit timer H1 (carrier generator mode).
Watchdog Timer	WDT_Restart	Clears the watchdog timer counter and resumes counting.
Real-time Clock	RTC_Init	Performs initialization necessary to control real-time counter functions.
	RTC_UserInit	Performs user-defined initialization relating to the real-time counter.
	RTC_PowerOff	Halts the clock supplied to the real-time counter.
	RTC_CounterEnable	Starts the count of the real-time counter (year, month, weekday, day, hour, minute, second).
	RTC_CounterDisable	Ends the count of the real-time counter (year, month, weekday, day, hour, minute, second).
	RTC_SetHourSystem	Sets the clock type (12-hour or 24-hour clock) of the real-time counter.
	RTC_CounterSet	Sets the counter value (year, month, weekday, day, hour, minute, second) of the real-time counter.
	RTC_CounterGet	Reads the counter value (year, month, weekday, day, hour, minute, second) of the real-time counter.
	RTC_ConstPeriodInterruptEnable	Sets the cycle of the interrupts INTRTC, then starts the cyclic interrupt function.
	RTC_ConstPeriodInterruptDisable	Ends the cyclic interrupt function.
	RTC_ConstPeriodInterruptCallback	Performs processing in response to the cyclic interrupt INTRTC.
	RTC_AlarmEnable	Starts the alarm interrupt function.
	RTC_AlarmDisable	Ends the alarm interrupt function.
	RTC_AlarmSet	Sets the alarm conditions (weekday, hour, minute).
	RTC_AlarmGet	Reads the alarm conditions (weekday, hour, minute).
	RTC_AlarmInterruptCallback	Performs processing in response to the alarm interrupt INTRTC.
	RTC_IntervalStart	Starts the interval interrupt function.
	RTC_IntervalStop	Ends the interval interrupt function.
	RTC_IntervalInterruptEnable	Sets the cycle of the interrupts, then starts the interval interrupt INTRTCI function.
	RTC_IntervalInterruptDisable	Ends the interval interrupt function.
	RTC_RTC1HZ_OutputEnable	Enables output of the real-time counter correction clock (1 Hz) to the RTC1HZ pin.

Peripheral Function	API Function Name	Function
Real-time Clock	RTC_RTC1HZ_OutputDisable	Disables output of the real-time counter correction clock (1 Hz) to the RTC1HZ pin.
	RTC_RTCCL_OutputEnable	Enables output of the real-time counter clock (32 kHz source) to the RTCCL pin.
	RTC_RTCCL_OutputDisable	Disables output of the real-time counter clock (32 kHz source) to the RTCCL pin.
	RTC_RTCDIV_OutputEnable	Enables output of the real-time counter clock (32 kHz cycle) to the RTCDIV pin.
	RTC_RTCDIV_OutputDisable	Disables output of the real-time counter clock (32 kHz cycle) to the RTCDIV pin.
	RTC_ChangeCorrectionValue	Changes the timing and correction value for correcting clock errors.
Clock Output	PCL_Init	Performs initialization necessary to control clock output control circuit functions.
	PCL_UserInit	Performs user-defined initialization relating to the clock output control circuits.
	PCL_Start	Starts clock output.
	PCL_Stop	Ends clock output.
	PCL_ChangeFreq	Changes the output clock to the PCL pin.
LVI	LVI_Init	Performs initialization necessary to control low-voltage detector functions.
	LVI_UserInit	Performs user-defined initialization relating to the low-voltage detector.
	LVI_InterruptModeStart	Starts low-voltage detection (when in interrupt generation mode).
	LVI_ResetModeStart	Starts low-voltage detection (when in internal reset mode).
	LVI_Stop	Stops low-voltage detection.
	LVI_SetLVILevel	Sets the low-voltage detection level.

C.3 Function Reference

This section describes the API functions output by Code Generator, using the following notation format.

Figure C-1. Notation Format of API Functions



(1) Name

Indicates the name of the API function.

(2) Outline

Outlines the functions of the API function.

(3) [Classification]

Indicates the name of the C source file to which the API function is output.

(4) [Syntax]

Indicates the format to be used when describing an API function to be called in C language.

(5) [Argument(s)]

API function arguments are explained in the following format.

I/O	Argument	Description
(a)	(b)	(c)

(a) I/O

Argument classification

I ... Input argument

O ... Output argument

(b) Argument

Argument data type

(c) Description

Description of argument

(6) [Return value]

API function return value is explained in the following format.

Macro	Description
(a)	(b)

(a) Macro

Macro of return value

(b) Description

Description of return value

(7) [Example]

Shows an example of the API function in use.

C.3.1 System

Below is a list of API functions output by Code Generator for system use.

Table C-2. API Functions: [System]

API Function Name	Function
CLOCK_Init	Performs initialization required to control the clock generator, on-chip debug, and etc. .
CLOCK_UserInit	Performs user-defined initialization relating to the clock generator, on-chip debug, and etc. .
CG_ReadResetSource	Performs processing in response to RESET signal.
CG_ChangeClockMode	Changes the CPU clock/peripheral hardware clock.
CG_ChangeFrequency	Changes the division ratio of the CPU clock/peripheral hardware clock.
CG_SelectPowerSaveMode	Configures the CPU's standby function.
CG_SelectStabTime	Configures the oscillation stabilization time of the X1 clock.
CG_ChangePllMode	Controls the operation of PLL function.

CLOCK_Init

Performs initialization required to control the clock generator, on-chip debug and etc. .

[Classification]

CG_system.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

CLOCK_UserInit

Performs user-defined initialization relating to the clock generator, on-chip debug, and etc. .

Remark This API function is called as the CLOCK_Init callback routine.

[Classification]

CG_system_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

CG_ReadResetSource

Performs processing in response to RESET signal.

[Classification]

CG_system_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

[Example]

Below are examples of the different processes executing depending on the RESET signal trigger.

[CG_Systeminit.c]

```
void systeminit ( void ) {
    CG_ReadResetSource ();      /* Processes executed by RESET signal trigger */
    ......
}
```

[CG_system_user.c]

```
#include    "CG_macrodriver.h"
void CG_ReadResetSource ( void ) {
    UCHAR   flag = RESF;        /* Reset control flag register: Obtain RESF contents */
    if ( flag & 0x1 ) {         /* Trigger identification: Check LVIRF flag */
        ......  /* Internal reset request by low-voltage detector */
    } else if ( flag & 0x10 ) { /* Trigger identification: Check WDTRF flag */
        ......  /* Internal reset request by watchdog timer */
    }
    ......
}
```

CG_ChangeClockMode

Changes the CPU clock/peripheral hardware clock.

[Classification]

CG_system.c

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	`enum ClockMode mode;`	CPU clock (fCPU)/peripheral hardware clock (fPRS) type HIOCLK: fCPU/fPRS >> Internal high-speed oscillation clock HIOSYSCLK: fCPU >> Internal high-speed oscillation clock, fPRS >> High-spped system clock SYSX1CLK: fCPU/fPRS >> X1 clock SYSEXTCLK: fCPU/fPRS >> External main system clock SUBXT1CLK: fCPU >> XT1 clock SUBEXTCLK: fCPU >> External subsystem clock

Remark SUBXT1CLK and SUBEXTCLK can only be specified when the target device is a 78K0/KC2-L.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Exit with error (abend) - Cannot change fCPU/fPRS to the internal high-speed osillation clock.
MD_ERROR2	Exit with error (abend) - Cannot change fCPU to the internal high-speed oscillation clock. - Cannot change fPRS to the high-speed system clock.
MD_ERROR3	Exit with error (abend) - Cannot change fCPU/fPRS to the X1 clock.
MD_ERROR4	Exit with error (abend) - Cannot change fCPU/fPRS to the external main system clock.
MD_ERROR5	Exit with error (abend) [Kx2-L] - Cannot change fCPU to the XT1 clock.
MD_ERROR6	Exit with error (abend) [Kx2-L] - Cannot change fCPU to the external subsystem clock.
MD_ARGERROR	Invalid argument specification

Remark The values MD_ERROR5 and MD_ERROR6 will only be returned when the target device is a 78K0/KC2-L.

CG_ChangeFrequency

Changes the division ratio of the CPU clock/peripheral hardware clock.

[Classification]

CG_system.c

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	`enum CPUClock clock;`	Division ratio type SYSTEMCLOCK: fMAIN SYSONEHALF: fMAIN/2 SYSONEFOURTH: fMAIN/4 SYSONEEIGHTH: fMAIN/8 SYSONESIXTEENTH: fMAIN/16

Remark "fMAIN" signifies the frequency of the main system clock.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

CG_SelectPowerSaveMode

Configures the CPU's standby function.

[Classification]

CG_system.c

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	`enum PSLevel level;`	Standby function type PSSTOP: STOP mode PSHALT: HALT mode

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR	Exit with error (abend) [Kx2-L] - If the CPU is operating by XT1 clock, then STOP mode cannot be specified.
MD_ARGERROR	Invalid argument specification

Remark The value MD_ERROR will only be returned when the target device is a 78K0/KC2-L.

[Example]

Below is an example of changing the standby function to "STOP mode".

[CG_main.c]

```
#include    "CG_macrodriver.h"
#include    "CG_system.h"
void main ( void ) {
    MD_STATUS   ret;
    ......
    TM00_Stop ();                               /* Stop count */
    ret = CG_SelectPowerSaveMode ( PSSTOP );    /* Change to STOP mode */
    if ( ret != MD_OK ) {
        while ( 1 );
    }
```

```
    TM00_Init ();                          /* Initialize TM00 */
    TM00_Start ();                         /* Start count */
    ......
}
```

CG_SelectStabTime

Configures the oscillation stabilization time of the X1 clock.

[Classification]

CG_system.c

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	`enum StabTime waittime;`	Oscillation stabilization time type STLEVEL0: 2^11/fx STLEVEL1: 2^13/fx STLEVEL2: 2^14/fx STLEVEL3: 2^15/fx STLEVEL4: 2^16/fx

Remark "fx" signifies the frequency of the X1 clock.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

CG_ChangePllMode

Controls the operation of PLL function.

[Classification]

CG_system.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_system.h"
MD_STATUS   CG_ChangePllMode ( enum PllMode pllmode );
```

[Argument(s)]

I/O	Argument	Description
I	`enum PllMode pllmode;`	Control of operation SYSPLLON: Enable operation SYSPLLOFF: Stop operation

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

C.3.2 Port

Below is a list of API functions output by Code Generator for port use.

Table C-3. API Functions: [Port]

API Function Name	Function
PORT_Init	Performs initialization necessary to control port functions.
PORT_UserInit	Performs user-defined initialization relating to the port.
PORT_ChangePmnInput	Switches the pin's I/O mode from output mode to input mode.
PORT_ChangePmnOutput	Switches the pin's I/O mode from input mode to output mode.

PORT_Init

Performs initialization necessary to control port functions.

[Classification]

CG_port.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

PORT_UserInit

Performs user-defined initialization relating to the port.

Remark This API function is called as the PORT_Init callback routine.

[Classification]

CG_port_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

PORT_ChangeP*mn*Input

Switches the pin's I/O mode from output mode to input mode.

[Classification]

CG_port.c

[Syntax]

The format for specifying this API function differs according to whether the target pin has built-in pull-up resistance/a SMBus input buffer.

- Built-in pull-up resistance: none; SMBus input buffer: none

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

- Built-in pull-up resistance: yes; SUBus input buffer: none

```
#include    "CG_macrodriver.h"
void    PORT_ChangePmnInput ( BOOL enablepu );
```

- Built-in pull-up resistance: yes; SMBus input buffer: yes

```
#include    "CG_macrodriver.h"
void    PORT_ChangePmnInput ( BOOL enablepu, BOOL enablesmbus );
```

Remark *mn* is the port number.

[Argument(s)]

I/O	Argument	Description
I	BOOL *enablepu*;	Built-in pull-up resistance used MD_TRUE: Yes MD_FALSE: No
I	BOOL *enablesmbus*;	Input buffer type MD_TRUE: SMBus input buffer MD_FALSE: Normal input buffer

[Return value]

None.

[Example 1]

Below is shown an example where pin P00 (built-in pull-up resistance: yes; SMBus input buffer: none) is changed as follows:

I/O mode type: Input mode
Built-in pull-up resistance used: Yes

[CG_main.c]

```
#include    "CG_macrodriver.h"
void main ( void ) {
    ......
    PORT_ChangeP00Input ( MD_TRUE );    /* Switch I/O mode */
    ......
}
```

[Example 2]

Below is shown an example where pin P00 (built-in pull-up resistance: yes; SMBus input buffer: none) is changed as follows:

I/O mode type: Input mode
Built-in pull-up resistance used: No

[CG_main.c]

```
#include    "CG_macrodriver.h"
void main ( void ) {
    ......
    PORT_ChangeP00Input ( MD_FALSE );   /* Switch I/O mode */
    ......
}
```

[Example 3]

Below is shown an example where pin P04 (built-in pull-up resistance: yes; SMBus input buffer: yes) is changed as follows:

I/O mode type: Input mode
Built-in pull-up resistance used: No
Input buffer type: SMBus input buffer

[CG_main.c]

```
#include    "CG_macrodriver.h"
void main ( void ) {
    ......
    PORT_ChangeP04Input ( MD_FALSE, MD_TRUE );  /* Switch I/O mode */
    ......
}
```

PORT_ChangeP*mn*Output

Switches the pin's I/O mode from input mode to output mode.

[Classification]

CG_port.c

[Syntax]

The format for specifying this API function differs according to whether the target pin conducts N-ch open drain output.

- N-ch open drain output: none

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

- N-ch open drain output: yes

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

Remark *nm* is the port number.

[Argument(s)]

I/O	Argument	Description
I	BOOL *enablench*;	Output mode type MD_TRUE: N-ch open drain output (V_{DD} withstand voltage) mode MD_FALSE: Normal output mode
I	BOOL *initialvalue*;	Initial output value MD_SET: Output HIGH level "1" MD_CLEAR: Output LOW level "0"

[Return value]

None.

[Example 1]

Below is shown an example where pin P00 (N-ch open drain output: none) is changed as follows:

I/O mode type: Output mode
Initial output value: Output HIGH level "1"

[CG_main.c]

```
#include    "CG_macrodriver.h"
void main ( void ) {
    ......
    PORT_ChangeP00Output ( MD_SET );    /* Switch I/O mode */
```

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

[Example 2]

Below is shown an example where pin P04 (N-ch open drain output: yes) is changed as follows:

I/O mode type: Output mode
Output mode type: N-ch open drain output (V_{DD} withstand voltage) mode
Initial output value: Output LOW level "0"

[CG_main.c]

```
#include    "CG_macrodriver.h"
void main ( void ) {
    ......
    PORT_ChangeP04Output ( MD_TRUE, MD_CLEAR ); /* Switch I/O mode */
    ......
}
```

C.3.3 Interrupt

Below is a list of API functions output by Code Generator for interrupt and key interrupt use.

Table C-4. API Functions: [Interrupt]

API Function Name	Function
INTP_Init	Performs initialization necessary to control the external interrupt INTP*n* functions.
INTP_UserInit	Performs user-defined initialization relating to the external interrupt INTP*n* functions.
KEY_Init	Performs initialization necessary to control the key interrupt INTKR functions.
KEY_UserInit	Performs user-defined initialization relating to the key interrupt INTKR functions.
INT_MaskableInterruptEnable	Disables/enables the acceptance of the maskable interrupts.
INTPn_Disable	Disables the acceptance of the maskable interrupts INTP*n* (external interrupt requests).
INTPn_Enable	Enables the acceptance of the maskable interrupts INTP*n* (external interrupt requests).
KEY_Disable	Disables the acceptance of the key interrupts INTKR.
KEY_Enable	Enables the acceptance of the key interrupts INTKR.

INTP_Init

Performs initialization necessary to control the external interrupt INTP*n* functions.

[Classification]

CG_int.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

INTP_UserInit

Performs user-defined initialization relating to the external interrupt INTP*n* functions.

Remark This API function is called as the INTP_Init callback routine.

[Classification]

CG_int_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

KEY_Init

Performs initialization necessary to control the key interrupt INTKR functions.

[Classification]

CG_int.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

KEY_UserInit

Performs user-defined initialization relating to the key interrupt INTKR functions.

Remark This API function is called as the KEY_Init callback routine.

[Classification]

CG_int_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

INT_MaskableInterruptEnable

Disables/enables the acceptance of the maskable interrupts.

[Classification]

CG_int.c

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	`enum MaskableSource` *`name`*`;`	Maskable interrupt type INT_*xxx*: Maskable interrupt
I	`BOOL` *`enableflag`*`;`	Acceptance enabled/disabled MD_TRUE: Acceptance enabled MD_FALSE: Acceptance disabled

Remark See the header file CG_int.h for details about the maskable interrupt type INT_*xxx*.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

[Example 1]

Below is an example of disabling acceptance of the maskable interrupt INTP0.

[CG_main.c]

```
#include    "CG_macrodriver.h"
#include    "CG_int.h"
void main ( void ) {
    ......
    INT_MaskableInterruptEnable ( INT_INTP0, MD_FALSE );  /* Disable acceptance of maskable
interrupt INTP0 */
    ......
}
```

[Example 2]

Below is an example of enabling acceptance of the maskable interrupt INTP0.

[CG_main.c]

```
#include    "CG_macrodriver.h"
#include    "CG_int.h"
void main ( void ) {
    ......
    INT_MaskableInterruptEnable ( INT_INTP0, MD_TRUE );   /* Enable acceptance of maskable
interrupt INTP0 */
    ......
}
```

INTP*n*_Disable

Disables the acceptance of the maskable interrupts INTP*n* (external interrupt requests).

[Classification]

CG_int.c

[Syntax]

```
void    INTPn_Disable ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

INTP*n*_Enable

Enables the acceptance of the maskable interrupts INTP*n* (external interrupt requests).

[Classification]

CG_int.c

[Syntax]

```
void    INTPn_Enable ( void );
```

Remark *n* is the interrupt factor number.

[Argument(s)]

None.

[Return value]

None.

KEY_Disable

Disables the acceptance of the key interrupts INTKR.

[Classification]

CG_int.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

KEY_Enable

Enables the acceptance of the key interrupts INTKR.

[Classification]

CG_int.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

C.3.4 Serial

Below is a list of API functions output by Code Generator for serial array unit and serial interface use.

Table C-5. API Functions: [Serial]

API Function Name	Function
UART6_Init	Performs initialization of the serial interface (UART6) channel.
UART6_UserInit	Performs user-defined initialization of the serial interface (UART6).
UART6_Start	Sets UART communication to standby mode.
UART6_Stop	Ends UART communication.
UART6_SendData	Starts UART data transmission.
UART6_ReceiveData	Starts UART data reception.
UART6_SendEndCallback	Performs processing in response to the UART transmission complete interrupt INTST6.
UART6_ReceiveEndCallback	Performs processing in response to the UART reception complete interrupt INTSR6.
UART6_SoftOverRunCallback	Performs processing in response to the serial transfer end interrupt INTSR6.
UART6_ErrorCallback	Performs processing in response to the UART communication error interrupt INTSRE6.
CSI1n_Init	Performs initialization of the serial interface (CSI1*n*) channel.
CSI1n_UserInit	Performs user-defined initialization of the serial interface (CSI1*n*).
CSI1n_Start	Sets CSI1*n* communication to standby mode.
CSI1n_Stop	Ends CSI1*n* communication.
CSI1n_ReceiveData	Starts CSI1*n* data reception.
CSI1n_SendReceiveData	Starts CSI1*n* data transmission/reception.
CSI1n_SendEndCallback	Performs processing in response to the CSI1*n* communication complete interrupt INTCSI1*n*.
CSI1n_ReceiveEndCallback	Performs processing in response to the CSI1*n* communication complete interrupt INTSCI1*n*.
IICA_Init	Performs initialization of the serial interface (IICA).
IICA_UserInit	Performs user-defined initialization of the serial interface (IICA).
IICA_Stop	Ends IICA communication.
IICA_MasterSendStart	Starts IICA master transmission.
IICA_MasterReceiveStart	Starts IICA master reception.
IICA_StopCondition	Generates stop conditions.
IICA_MasterSendEndCallback	Performs processing in response to the IICA communication complete interrupt INTIICA0.
IICA_MasterReceiveEndCallback	Performs processing in response to the IICA communication complete interrupt INTIICA0.
IICA_MasterErrorCallback	Performs processing in response to detection of error in IICA master communication.
IICA_SlaveSendStart	Starts IICA slave transmission.
IICA_SlaveReceiveStart	Starts IICA slave reception.

API Function Name	Function
IICA_SlaveSendEndCallback	Performs processing in response to the IICA communication complete interrupt INTIICA0.
IICA_SlaveReceiveEndCallback	Performs processing in response to the IICA communication complete interrupt INTIICA0.
IICA_SlaveErrorCallback	Performs processing in response to detection of error in IICA slave communication.
IICA_GetStopConditionCallback	Performs processing in response to detection of stop condition in IICA slave communication.

UART6_Init

Performs initialization of the serial interface (UART6) channel.

[Classification]

CG_serial.c

[Syntax]

```
void    UART6_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

UART6_UserInit

Performs user-defined initialization of the serial interface (UART6).

Remark This API function is called as the UART6_Init callback routine.

[Classification]

CG_serial_user.c

[Syntax]

```
void    UART6_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

UART6_Start

Sets UART communication to standby mode.

[Classification]

CG_serial.c

[Syntax]

```
void    UART6_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

UART6_Stop

Ends UART communication.

[Classification]

CG_serial.c

[Syntax]

```
void    UART6_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

UART6_SendData

Starts UART data transmission.

Remarks 1. This API function repeats the byte-level UART transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.

2. When performing a UART transmission, UART6_Start must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   UART6_SendData ( UCHAR *txbuf, USHORT txnum );
```

[Argument(s)]

I/O	Argument	Description
I	`UCHAR *txbuf;`	Pointer to a buffer storing the transmission data
I	`USHORT txnum;`	Total amount of data to send

Remark You can only set 1 for the total *txnum* of the sending data in case the serial interface (UART6) operates in DALI mode.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

[Example]

Below is an example of sending a UART transmission of four bytes of fixed-length data one time.

[CG_main.c]

```
#include    "CG_macrodriver.h"
BOOL    gFlag;                          /* Transmission complete flag */
void main ( void ) {
    UCHAR   txbuf[] = "ABCD";
    USHORT  txnum = 4;
    gFlag = 1;                          /* Initialize transmission complete flag */
    ......
    UART6_Start ();                     /* Start UART communication*/
    UART6_SendData ( &txbuf, txnum );   /* Start UART data transmission */
```

```
    while ( gFlag );                    /* Wait for txnum transmissions */
    ......
}
```

[CG_serial_user.c]

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                  /* Transmission complete flag */
__interrupt void MD_INTST6 ( void ) {   /* Interrupt processing for INTST6 */
    if ( gUart6TxCnt > 0 ) {
        ......
    } else {
        UART6_SendEndCallback ();       /* Call callback routine */
    }
}

void UART6_SendEndCallback ( void ) {   /* Callback routine for INTST6 */
    gFlag = 0;                          /* Set transmission complete flag */
}
```

UART6_ReceiveData

Starts UART data reception.

Remarks 1. This API function performs byte-level UART reception the number of times specified by the parameter *rxnum*, and stores the data in the buffer specified by the parameter *rxbuf*.

2. Actual UART reception starts after this API function is called, and UART6_Start is then called.

[Classification]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   UART6_ReceiveData ( UCHAR *rxbuf, USHORT rxnum );
```

[Argument(s)]

I/O	Argument	Description
O	`UCHAR *rxbuf;`	Pointer to a buffer to store the received data
I	`USHORT rxnum;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

Remark You can only set 2 for the total *rxnum* of the receiving data in case the serial interface (UART6) operates in DALI mode.

[Example]

Below is an example of UART reception of four bytes of fixed-length data one time.

[CG_main.c]

```
#include    "CG_macrodriver.h"
BOOL    gFlag;                          /* Reception complete flag */
void main ( void ) {
    UCHAR   rxbuf[10];
    USHORT  rxnum = 4;
    gFlag = 1;                          /* Initialize reception complete flag */
    ......
    UART6_ReceiveData ( &rxbuf, rxnum );    /* Start UART data reception */
    UART6_Start ();                     /* Start UART communication */
```

```
    while ( gFlag );                        /* Wait for rxnum receptions */
    ......
}
```

[CG_serial_user.c]

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                      /* Reception complete flag */
__interrupt void MD_INTSR6 ( void ) {       /* Interrupt processing for INTSR6 */
    ......
    if ( gUart6RxLen > gUart6RxCnt ) {
        ......
        if ( gUart6RxLen == gUart6RxCnt ) {
            UART6_ReceiveEndCallback ();    /* Call callback routine */
        }
    }
}

void UART6_ReceiveEndCallback ( void ) {    /* Callback routine for INTSR6 */
    gFlag = 0;                              /* Set reception complete flag */
}
```

UART6_SendEndCallback

Performs processing in response to the UART transmission complete interrupt INTST6.

Remark This API function is called as the callback routine of interrupt process MD_INTST6 corresponding to the UART transmission complete interrupt INTST6 (performed when number of transmission data specified by UART6_SendData parameter *txnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void    UART6_SendEndCallback ( void );
```

[Argument(s)]

None.

[Return value]

None.

UART6_ReceiveEndCallback

Performs processing in response to the UART reception complete interrupt INTSR6.

Remark This API function is called as the callback routine of interrupt process MD_INTSR6 corresponding to the UART reception complete interrupt INTSR6 (performed when number of received data specified by UART6_ReceiveData parameter *rxnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void    UART6_ReceiveEndCallback ( void );
```

[Argument(s)]

None.

[Return value]

None.

UART6_SoftOverRunCallback

Performs processing in response to the UART reception complete interrupt INTSR6.

Remark This API function is called as the callback routine of interrupt process MD_INTSR6 corresponding to the UART reception complete interrupt INTSR6 (process performed when the amount of data received is greater than the parameter *rxnum* specified for UART6_ReceiveData).

[Classification]

CG_serial_user.c

[Syntax]

- [Ix2]

```
void    UART6_SoftOverRunCallback ( USHORT rx_data );
```

- [Kx2-L]

```
void    UART6_SoftOverRunCallback ( UCHAR rx_data );
```

[Argument(s)]

I/O	Argument	Description
O	`USHORT rx_data;`	Received data (data received greater than the number specified in the parameter *rxnum* for UART6_ReceiveData)
O	`UCHAR rx_data;`	Received data (data received greater than the number specified in the parameter *rxnum* for UART6_ReceiveData)

[Return value]

None.

UART6_ErrorCallback

Performs processing in response to the UART communication error interrupt INTSRE6.

Remark This API function is called as the callback routine of interrupt process MD_INTSRE6 corresponding to the UART communication error interrupt INTSRE6.

[Classification]

CG_serial_user.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    UART6_ErrorCallback ( UCHAR err_type );
```

[Argument(s)]

I/O	Argument	Description
O	UCHAR *err_type*;	Trigger for UART communication error interrupt 00000*xx*1B: Overrun error 00000*x*1*x*B: Framing error 000001*xx*B: Parity error

[Return value]

None.

[Example]

Below are examples of callback processing by the trigger for the UART communication error interrupt.

[CG_serial_user.c]

```
#include    "CG_macrodriver.h"
__interrupt void MD_INTSRE6 ( void ) {          /* Interrupt processing for INTSRE6 */
    UCHAR   err_type;
    ......
    UART6_ErrorCallback ( err_type );           /* Call callback routine */
}

void UART6_ErrorCallback ( UCHAR err_type ) {   /* Callback routine for INTSRE6 */
    if ( err_type & 0x1 ) {                     /* Determine trigger */
        ......  /* Callback processing in response to overrun error */
    } else if ( err_type & 0x2 ) {              /* Determine trigger */
        ......  /* Callback processing in response to framing error */
    } else if ( err_type & 0x4 ) {              /* Determine trigger */
        ......  /* Callback processing in response to parity error */
```

```
    }
}
```

CSI1*n*_Init

Performs initialization of the serial interface (CSI1*n*) channel.

[Classification]

CG_serial.c

[Syntax]

```
void    CSI1n_Init ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSI1*n*_UserInit

Performs user-defined initialization of the serial interface (CSI1*n*).

Remark This API function is called as the CSI1n_Init callback routine.

[Classification]

CG_serial_user.c

[Syntax]

```
void    CSI1n_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSI1*n*_Start

Sets CSI1*n* communication to standby mode.

[Classification]

CG_serial.c

[Syntax]

```
void    CSI1n_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSI1*n*_Stop

Ends CSI1*n* communication.

[Classification]

CG_serial.c

[Syntax]

```
void    CSI1n_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSI1*n*_ReceiveData

Starts CSI1*n* data reception.

Remarks 1. This API function performs byte-level CSI1*n* reception the number of times specified by the parameter *rxnum*, and stores the data in the buffer specified by the parameter *rxbuf*.

2. When performing a CSI1*n* reception, CSI1n_Start must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   CSI1n_ReceiveData ( UCHAR *rxbuf, USHORT rxnum );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
O	`UCHAR *rxbuf;`	Pointer to a buffer to store the received data
I	`USHORT rxnum;`	Total amount of data to receive

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

[Example]

Below is an example of receiving a CSI10 transmission of four bytes of fixed-length data from channel 10 one time.

[CG_main.c]

```
#include    "CG_macrodriver.h"
BOOL    gFlag;                          /* Reception complete flag */
void main ( void ) {
    UCHAR   rxbuf[10];
    USHORT  rxnum = 4;
    gFlag = 1;                          /* Initialize reception complete flag */
    ......
    CSI10_Start ();                     /* Start CSI10 communication */
    CSI10_ReceiveData ( &rxbuf, rxnum );    /* Start CSI10 reception */
    while ( gFlag );                    /* Wait for rxnum receptions */
```

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

[CG_serial_user.c]

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                      /* Reception complete flag */
__interrupt void MD_INTCSI10 ( void ) {     /* Interrupt processing for INTCSI10 */
    if ( gCsi10RxCnt < gCsi10RxLen ) {
        ......
        if ( gCsi10RxCnt == gCsi10RxLen ) {
            CSI10_ReceiveEndCallback ();    /* Call callback routine */
        } else {
            ......
        }
    }
}

void CSI10_ReceiveEndCallback ( void ) {    /* Callback routine for INTCSI10 */
    gFlag = 0;                              /* Set reception complete flag */
}
```

CSI1*n*_SendReceiveData

Starts CSI1*n* data transmission/reception.

Remarks 1. This API function repeats the byte-level CSI1*n* transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.

2. This API function performs byte-level CSI1*n* reception the number of times specified by the parameter *txnum*, and stores the data in the buffer specified by the parameter *rxbuf*.

3. When performing a CSI1*n* reception, CSI1n_Start must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

```
#include    "CG_macrodriver.h"
MD_STATUS   CSI1n_SendReceiveData ( UCHAR *txbuf, USHORT txnum, UCHAR *rxbuf );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	UCHAR *txbuf;	Pointer to a buffer storing the transmission data
I	USHORT txnum;	Total amount of data to send/receive
O	UCHAR *rxbuf;	Pointer to a buffer to store the received data

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

[Example]

Below is an example of sending and receiving a CSI10 transmission of four bytes of fixed-length data from channel 10 one time.

[CG_main.c]

```
#include    "CG_macrodriver.h"
BOOL    gSflag;                                     /* Transmission complete flag */
void main ( void ) {
    UCHAR   txbuf[] = "0123";
    USHORT  txnum = 4;
    UCHAR   rxbuf[10];
```

```
    gSflag = 1;                                         /* Initialize flag */
    ......
    CSI10_Start ();                                     /* Start CSI10 communication */
    CSI10_SendReceiveData ( &txbuf, txnum, &rxbuf );    /* Start CSI10 send/receive */
   while ( gSlag );                                  /* Wait for txnum transmissions/receptions
*/
    ......
}
```

[CG_serial_user.c]

```
#include    "CG_macrodriver.h"
extern  BOOL    gSflag;                     /* Transmission complete flag */
__interrupt void MD_INTCSI10 ( void ) {     /* Interrupt processing for INTCSI10 */
   if ( gCsi10TxCnt > 0 ) {
        ......
    } else {
        ......
        CSI10_SendEndCallback ();           /* Call callback routine */
    }
    ......
}

void CSI10_SendEndCallback ( void ) {       /* Callback routine for INTCSI10 */
    gSflag = 0;                             /* Set transmission complete flag */
}
```

CSI1*n*_SendEndCallback

Performs processing in response to the CSI1*n* communication complete interrupt INTCSI1*n*.

Remark This API function is called as the callback routine of interrupt process MD_INTCSI1*n* corresponding to the CSI1*n* communication complete interrupt INTCSI1*n* (performed when number of transmission data specified by CSI1n_SendReceiveData parameter *txnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void    CSI1n_SendEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSI1*n*_ReceiveEndCallback

Performs processing in response to the CSI1*n* communication complete interrupt INTCSI1*n*.

Remark This API function is called as the callback routine of interrupt process MD_INTCSI1*n* corresponding to the CSI1*n* communication complete interrupt INTCSI1*n* (performed when number of received data specified by CSI1n_ReceiveData parameter *rxnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void    CSI1n_ReceiveEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IICA_Init

Performs initialization of the serial interface (IICA).

[Classification]

CG_serial.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

IICA_UserInit

Performs user-defined initialization of the serial interface (IICA).

Remark This API function is called as the IICA_Init callback routine.

[Classification]

CG_serial_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

IICA_Stop

Ends IICA communication.

[Classification]

CG_serial.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

IICA_MasterSendStart

Starts IICA master transmission.

Remark This API function repeats the byte-level IICA master transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.

[Classification]

CG_serial.c

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	UCHAR *adr*;	Slave address
I	UCHAR **txbuf*;	Pointer to a buffer storing the transmission data
I	USHORT *txnum*;	Total amount of data to send
I	UCHAR *wait*;	Setup time of start conditions

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Bus communication status
MD_ERROR2	Bus not released status

IICA_MasterReceiveStart

Starts IICA master reception.

Remark This API function performs byte-level IICA master reception the number of times specified by the parameter *rxnum*, and stores the data in the buffer specified by the parameter *rxbuf*.

[Classification]

CG_serial.c

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	`UCHAR adr;`	Slave address
O	`UCHAR *rxbuf;`	Pointer to a buffer to store the received data
I	`USHORT rxnum;`	Total amount of data to receive
I	`UCHAR wait;`	Setup time of start conditions

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR1	Bus communication status
MD_ERROR2	Bus not released status

IICA_StopCondition

Generates stop conditions.

[Classification]

CG_serial.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

IICA_MasterSendEndCallback

Performs processing in response to the IICA communication complete interrupt INTIICA0.

Remark This API function is called as the callback routine of interrupt process MD_INTIICA0 corresponding to the IICA communication complete interrupt INTIICA0.

[Classification]

CG_serial_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

IICA_MasterReceiveEndCallback

Performs processing in response to the IICA communication complete interrupt INTIICA0.

Remark This API function is called as the callback routine of interrupt process MD_INTIICA0 corresponding to the IICA communication complete interrupt INTIICA0.

[Classification]

CG_serial_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

IICA_MasterErrorCallback

Performs processing in response to detection of error in IICA master communication.

[Classification]

CG_serial_user.c

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	`MD_STATUS flag;`	Cause of communication error MD_SPT: Stop condition detected MD_NACK: Acknowledge not detected

[Return value]

None.

IICA_SlaveSendStart

Starts IICA slave transmission.

Remark This API function repeats the byte-level IICA slave transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.

[Classification]

CG_serial.c

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	`UCHAR *txbuf;`	Pointer to a buffer storing the transmission data
I	`USHORT txnum;`	Total amount of data to send

[Return value]

None.

IICA_SlaveReceiveStart

Starts IICA slave reception.

Remark This API function performs byte-level IICA slave reception the number of times specified by the parameter *rxnum*, and stores the data in the buffer specified by the parameter *rxbuf*.

[Classification]

CG_serial.c

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
O	`UCHAR *rxbuf;`	Pointer to a buffer to store the received data
I	`USHORT rxnum;`	Total amount of data to receive

[Return value]

None.

IICA_SlaveSendEndCallback

Performs processing in response to the IICA communication complete interrupt INTIICA0.

Remark This API function is called as the callback routine of interrupt process MD_INTIICA0 corresponding to the IICA communication complete interrupt INTIICA0.

[Classification]

CG_serial_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

IICA_SlaveReceiveEndCallback

Performs processing in response to the IICA communication complete interrupt INTIICA0.

Remark This API function is called as the callback routine of interrupt process MD_INTIICA0 corresponding to the IICA communication complete interrupt INTIICA0.

[Classification]

CG_serial_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

IICA_SlaveErrorCallback

Performs processing in response to detection of error in IICA slave communication.

[Classification]

CG_serial_user.c

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	`MD_STATUS flag;`	Cause of communication error MD_ERROR: Address mismatch detected MD_NACK: Acknowledge not detected

[Return value]

None.

IICA_GetStopConditionCallback

Performs processing in response to detection of stop condition in IICA slave communication.

[Classification]

CG_serial_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

C.3.5 Operational Amplifier

Below is a list of API functions output by Code Generator for operational amplifiers use.

Table C-6. API Functions: [Operational Amplifier]

API Function Name	Function
OPAMP_Init	Performs initialization necessary to control operational amplifier functions.
OPAMP_UserInit	Performs user-defined initialization relating to the operational amplifier.
PGA_Start	Starts the operation of opeational amplifier (PGA mode).
PGA_Stop	Ends the operation of operational amplifier (PGA mode).
PGA_ChangePGAFactor	Sets the input voltage amplification factor of a operational amplifier (PGA mode).
AMP_Start	Starts the operation of opeational amplifier (single AMP mode).
AMP_Stop	Ends the operation of operational amplifier (single AMP mode).
AMPn_Start	Starts the operation of opeational amplifier *n* (single AMP mode).
AMPn_Stop	Ends the operation of operational amplifier *n* (single AMP mode).

OPAMP_Init

Performs initialization necessary to control operational amplifier functions.

[Classification]

CG_opamp.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

OPAMP_UserInit

Performs user-defined initialization relating to the operational amplifier.

Remark This API function is called as the OPAMP_Init callback routine.

[Classification]

CG_opamp_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

PGA_Start

Starts the operation of operational amplifier (PGA mode).

[Classification]

CG_opamp.c

[Syntax]

```
void    PGA_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

PGA_Stop

Ends the operation of operational amplifier (PGA mode).

[Classification]

CG_opamp.c

[Syntax]

```
void    PGA_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

PGA_ChangePGAFactor

Sets the input voltage amplification factor of a operational amplifier (PGA mode).

Remark The value specified in parameter *factor* is set to operational amplifier control register (AMP0M).

[Classification]

CG_opamp.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_opamp.h"
MD_STATUS   PGA_ChangePGAFactor ( enum PGAFactor factor );
```

[Argument(s)]

I/O	Argument	Description
I	`enum PGAFactor factor;`	Input voltage amplification factor PGAFACTOR0: x4 PGAFACTOR1: x8 PGAFACTOR2: x16 PGAFACTOR3: x32

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

AMP_Start

Starts the operation of operational amplifier (single AMP mode).

[Classification]

CG_opamp.c

[Syntax]

```
void    AMP_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

AMP_Stop

Ends the operation of operational amplifier (single AMP mode).

[Classification]

CG_opamp.c

[Syntax]

```
void    AMP_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

AMP*n*_Start

Starts the operation of operational amplifier *n* (single AMP mode).

[Classification]

CG_opamp.c

[Syntax]

```
void    AMPn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

AMP*n*_Stop

Ends the operation of operational amplifier *n* (single AMP mode).

[Classification]

CG_opamp.c

[Syntax]

```
void    AMPn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

C.3.6 Comparator

Below is a list of API functions output by Code Generator for comparator use.

Table C-7. API Functions: [Comparator]

API Function Name	Function
Comparator_Init	Performs initialization necessary to control comparator functions.
Comparator_UserInit	Performs user-defined initialization relating to the comparator.
Comparatorn_Start	Starts the operation of comparator *n*.
Comparatorn_Stop	Ends the operation of comparator *n*.

Comparator_Init

Performs initialization necessary to control comparator functions.

[Classification]

CG_comparator.c

[Syntax]

```
void    Comparator_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

Comparator_UserInit

Performs user-defined initialization relating to the comparator.

Remark This API function is called as the Comparator_Init callback routine.

[Classification]

CG_comparator_user.c

[Syntax]

```
void    Comparator_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

Comparator*n*_Start

Starts the operation of comparator *n*.

[Classification]

CG_comparator.c

[Syntax]

```
void    Comparatorn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

Comparator*n*_Stop

Ends the operation of comparator.

[Classification]

CG_comparatorn.c

[Syntax]

```
void    Comparator n_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

C.3.7 A/D Converter

Below is a list of API functions output by Code Generator for A/D converter use.

Table C-8. API Functions: [A/D Converter]

API Function Name	Function
AD_Init	Performs initialization necessary to control A/D converter functions.
AD_UserInit	Performs user-defined initialization relating to the A/D converter.
AD_ComparatorOn	Enables operation of voltage converter.
AD_ComparatorOff	Disables operation of voltage converter.
AD_Start	Starts A/D conversion.
AD_Stop	Ends A/D conversion.
AD_SelectADChannel	Configures the analog voltage input pin for A/D conversion.
AD_Read	Reads the results of A/D conversion (10 bits).
AD_ReadByte	Reads the results of A/D conversion (8 bits; most significant 8 bits of 10-bit resolution).

AD_Init

Performs initialization necessary to control A/D converter functions.

[Classification]

CG_ad.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

AD_UserInit

Performs user-defined initialization relating to the A/D converter.

Remark This API function is called as the AD_Init callback routine.

[Classification]

CG_ad_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

AD_ComparatorOn

Enables operation of voltage converter.

Remark About 1 microsecond of stabilization time is required when changing the voltage converter from operation stopped to operation enabled status.
Consequently, about 1 micro second must be left free between the call to this API function and the call to AD_Start.

[Classification]

CG_ad.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

AD_ComparatorOff

Disables operation of voltage converter.

[Classification]

CG_ad.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

AD_Start

Starts A/D conversion.

Remark About 1 micro second of stabilization time is required when changing the voltage converter from operation stopped to operation enabled status.
Consequently, about 1 micro second must be left free between the call to AD_ComparatorOn and the call to this API function.

[Classification]

CG_ad.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

AD_Stop

Ends A/D conversion.

Remark The voltage converter continues to operate after the process of this API function completes. Consequently, to stop the operation of the voltage converter, you must call AD_ComparatorOff after the process of this API function completes.

[Classification]

CG_ad.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

AD_SelectADChannel

Configures the analog voltage input pin for A/D conversion.

Remark The value specified in parameter *channel* is set to analog input channel specification register (ADS).

[Classification]

CG_ad.c

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	`enum ADChannel channel;`	Analog voltage input pin ADCHANNEL*n*: Input pin ADCHANNELPGAIN: Operational amplifier output pin ADCHANNEL12V: Internal voltage (1.2 V)

Remark See the header file CG_ad.h for details about the analog voltage input pin ADCHANNEL*n*.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

AD_Read

Reads the results of A/D conversion (10 bits).

Remark The contents of the 10-bit A/D conversion result register (ADCR) are stored in the area specified by parameter *buffer*.

[Classification]

CG_ad.c

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
O	`USHORT *buffer;`	Pointer to area in which to store read results of A/D conversion

[Return value]

None.

AD_ReadByte

Reads the results of A/D conversion (8 bits; most significant 8 bits of 10-bit resolution).

Remark The contents of the 8-bit A/D conversion result register H (ADCRH) are stored in the area specified by parameter *buffer*.

[Classification]

CG_ad.c

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
O	`UCHAR *buffer;`	Pointer to area in which to store the results of A/D conversion

[Return value]

None.

C.3.8 Timer

Below is a list of API functions output by Code Generator for timer array unit use.

Table C-9. API Functions: [Timer]

API Function Name	Function
TMX_Init	Performs initialization necessary to control 16-bit timer X*n* functions.
TMXn_Start	Starts the count for 16-bit timer X*n*.
TMXn_Stop	Ends the count for 16-bit timer X*n*.
TMXn_ChangeDuty	Changes the duty ratio of the PWM signal single-output to the TOX0*n* pin.
TMXn_ChangeDualDuty	Changes the duty ratio of the PWM signal dulal-output to the TOX0*n* pin.
TMX_EnableHighImpedanceState	Begins high impedance output of the 16-bit timer X*n*.
TMX_DisableHighImpedanceState	Ends high impedance output of the 16-bit timer X*n*.
TM00_Init	Performs initialization necessary to control 16-bit timer/event counter 00 functions.
TM00_UserInit	Performs user-defined initialization relating to the 16-bit timer/event counter 00.
TM00_Start	Starts the count for 16-bit timer/event counter 00.
TM00_Stop	Ends the count for 16-bit timer/event counter 00.
TM00_ChangeTimerCondition	Changes the value of capture/compare control register 00 (CRC00).
TM00_GetFreeRunningValue	Captures the content of the capture register (CR0*n*0).
TM00_SoftwareTriggerOn	Generates the trigger (software trigger) for one-shot pulse output.
TM00_ChangeDuty	Changes the duty ratio of the signal output to the TO00 pin.
TM00_GetPulseWidth	Captures the high/low-level width measured for the signal (pulses) input to the TI0*n*0 pin.
TM5n_Init	Performs initialization necessary to control 8-bit timer/event counter 5*n* functions.
TM5n_UserInit	Performs user-defined initialization relating to the 8-bit timer/event counter 5*n*.
TM5n_Start	Starts the count for 8-bit timer/event counter 5*n*.
TM5n_Stop	Ends the count for 8-bit timer/event counter 5*n*.
TM5n_ChangeTimerCondition	Changes the value of 8-bit timer compare register 5*n* (CR5*n*).
TM5n_ChangeDuty	Changes the duty ratio of the PWM signal output to the TO5*n* pin.
TMHn_Init	Performs initialization necessary to control 8-bit timer H*n* functions.
TMHn_UserInit	Performs user-defined initialization relating to the 8-bit timer H*n*.
TMHn_Start	Starts the count for 8-bit timer H*n*.
TMHn_Stop	Ends the count for 8-bit timer H*n*.
TMHn_ChangeTimerCondition	Changes the value of 8-bit timer H compare register 0*n*/1*n* (CMP0*n*/CMP1*n*).
TMHn_ChangeDuty	Changes the duty ratio of the PWM signal output to the TOH*n* pin.
TMH1_CarrierOutputEnable	Begins carrier pulse output of the 8-bit timer H1 (carrier generator mode).
TMH1_CarrierOutputDisable	Ends carrier pulse output of the 8-bit timer H1 (carrier generator mode).

TMX_Init

Performs initialization necessary to control 16-bit timer X*n* functions.

[Classification]

CG_timer.c

[Syntax]

```
void    TMX_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

TMX*n*_Start

Starts the count for 16-bit timer X*n*.

Remark The time from the call to this API function to the start of counting depends on the type of the function in question (e.g. PWMoutput, or A/D conversion start timing signal output).

[Classification]

CG_timer.c

[Syntax]

```
void    TMXn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMX*n*_Stop

Ends the count for 16-bit timer X*n*.

[Classification]

CG_timer.c

[Syntax]

```
void    TMXn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMX*n*_ChangeDuty

Changes the duty ratio of the PWM signal single-output to the TOX*n* pin.

Remark This API function can only be called when the 16-bit timer X*n* is being used for single-output.

[Classification]

CG_timer.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	UCHAR *ratio*;	Duty ratio (0 to 100, unit: %)

Remark The value set to duty ratio *ratio* must be in base 10 notation.

[Return value]

None.

[Example]

The example below shows changing the duty ratio to 25%.

[CG_main.c]

```
#include    "CG_macrodriver.h"
void main ( void ) {
    UCHAR   ratio = 25;
    ......
    TMX0_Start ();                  /* Start count */
    ......
    TMX0_ChangeDuty ( ratio );      /* Change duty ratio */
    ......
}
```

TMX*n*_ChangeDualDuty

Changes the duty ratio of the PWM signal dual-output to the TOX*n* pin.

Remark This API function can only be called when the 16-bit timer X*n* is being used for dual-output.

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    TMXn_ChangeDualDuty ( UCHAR ratio, UCHAR ratio1, UCHAR delay );
```

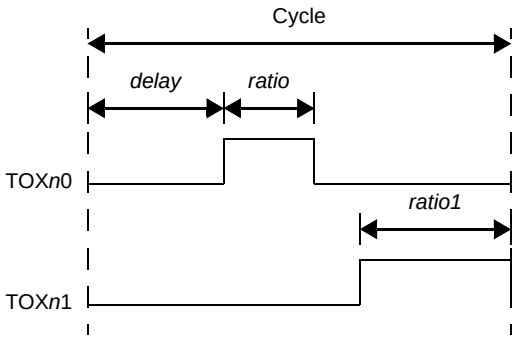
Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`UCHAR ratio;`	Duty ratio of TOX*n*0 (0 to 100, unit: %)
I	`UCHAR ratio1;`	Duty ratio of TOX*n*1 (0 to 100, unit: %)
I	`UCHAR delay;`	Delay time of TOX*n*0 (0 to 100, unit: %)

Remarks 1. The value set to duty ratio *ratio*, *ratio1* and *delay* must be in base 10 notation.

2. The following figure displays the meaning of each argument.



[Return value]

None.

TMX_EnableHighImpedanceState

Begins high impedance output of the 16-bit timer X*n*.

[Classification]

CG_timer.c

[Syntax]

```
void    TMX_EnableHighImpedanceState ( void );
```

[Argument(s)]

None.

[Return value]

None.

TMX_DisableHighImpedanceState

Ends high impedance output of the 16-bit timer X*n*.

[Classification]

CG_timer.c

[Syntax]

```
void    TMX_DisableHighImpedanceState ( void );
```

[Argument(s)]

None.

[Return value]

None.

TM00_Init

Performs initialization necessary to control 16-bit timer/event counter 00 functions.

[Classification]

CG_timer.c

[Syntax]

```
void    TM00_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

TM00_UserInit

Performs user-defined initialization relating to the 16-bit timer/event counter 00.

Remark This API function is called as the TM00_Init callback routine.

[Classification]

CG_timer_user.c

[Syntax]

```
void    TM00_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

TM00_Start

Starts the count for 16-bit timer/event counter 00.

Remark The time from the call to this API function to the start of counting depends on the type of the function in question (e.g. interval timer, square-wave output, or external event counter).

[Classification]

CG_timer.c

[Syntax]

```
void    TM00_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

TM00_Stop

Ends the count for 16-bit timer/event counter 00.

[Classification]

CG_timer.c

[Syntax]

```
void    TM00_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

TM00_ChangeTimerCondition

Changes the value of 16-bit timer capture/compare control register 0*n*0 (CR0*n*0).

Remark To change the contents of CR0*n*0, you must call TM00_Stop before calling this API function.

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_timer.h"
MD_STATUS   TM00_ChangeTimerCondition ( USHORT *array_reg, USHORT array_num );
```

[Argument(s)]

I/O	Argument	Description
I	`USHORT *array_reg;`	Pointer to the area storing the value to set in the target register
I	`USHORT array_num;`	The register to change 1: CR000 2: CR000, CR010

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR	Invalid argument "*array_num*" specification

TM00_GetFreeRunningValue

Captures the content of the capture register (CR0*n*0).

Remark This API function can only be called when the 16-bit timer/event counter 00 is running in free-running timer mode, and the 16-bit timer capture/compare control register 0*n*0 (CR0*n*0) is being used as a capture register.

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_timer.h"
MD_STATUS   TM00_GetFreeRunningValue ( ULONG *count, enum TMChannel channel );
```

[Argument(s)]

I/O	Argument	Description
I	`ULONG *count;`	Pointer to area in which to store the captured value
I	`enum TMChannel channel;`	The pin to capture TMCHANNEL0: TI000 pin TMCHANNEL1: TI010 pin

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR	Exit with error (abend) - CR0*n*0 is operating as a compare register.
MD_ARGERROR	Invalid argument specification

TM00_SoftwareTriggerOn

Generates the trigger (software trigger) for one-shot pulse output.

Remark This API function can only be called when the 16-bit timer/event counter 00 is being used for one-shot pulse output.

[Classification]

CG_timer.c

[Syntax]

```
void    TM00_SoftwareTriggerOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

TM00_ChangeDuty

Changes the duty ratio of the signal output to the TO00 pin.

Remark This API function can only be called when the 16-bit timer/event counter 00 is being used for PPG output.

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    TM00_ChangeDuty ( UCHAR ratio );
```

[Argument(s)]

I/O	Argument	Description
I	UCHAR *ratio*;	Duty ratio (0 to 100, unit: %)

Remark The value set to duty ratio *ratio* must be in base 10 notation.

[Return value]

None.

[Example]

The example below shows changing the duty ratio to 25%.

[CG_main.c]

```
#include    "CG_macrodriver.h"
void main ( void ) {
    UCHAR   ratio = 25;
    ......
    TM00_Start ();                  /* Start count */
    ......
    TM00_ChangeDuty ( ratio );      /* Change duty ratio */
    ......
}
```

TM00_GetPulseWidth

Captures the high/low-level width measured for the signal (pulses) input to the TI0*n*0 pin.

Remark This API function can only be called when the 16-bit timer/event counter 00 is being used for pulse width measurement.

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_timer.h"
void    TM00_GetPulseWidth ( ULONG *highwidth, ULONG *lowwidth, enum TMChannel channel );
```

[Argument(s)]

I/O	Argument	Description
O	`ULONG *highwidth;`	Pointer to area storing the high-level measurement width (0x0 to 0xffff)
O	`ULONG *lowwidth;`	Pointer to area storing the low-level measurement width (0x0 to 0xffff)
I	`enum TMChannel channel;`	The pin to measure TMCHANNEL0: TI000 pin TMCHANNEL1: TI010 pin

[Return value]

None.

TM5*n*_Init

Performs initialization necessary to control 8-bit timer/event counter 5*n* functions.

[Classification]

CG_timer.c

[Syntax]

```
void    TM5n_Init ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TM5*n*_UserInit

Performs user-defined initialization relating to the 8-bit timer/event counter 5*n*.

Remark This API function is called as the TM5n_Init callback routine.

[Classification]

CG_timer_user.c

[Syntax]

```
void    TM5n_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TM5*n*_Start

Starts the count for 8-bit timer/event counter 5*n*.

Remark The time from the call to this API function to the start of counting depends on the type of the function in question (e.g. interval timer, or external event counter).

[Classification]

CG_timer.c

[Syntax]

```
void    TM5n_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TM5*n*_Stop

Ends the count for 8-bit timer/event counter 5*n*.

[Classification]

CG_timer.c

[Syntax]

```
void    TM5n_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TM5*n*_ChangeTimerCondition

Changes the value of 8-bit timer compare register 5*n* (CR5*n*).

Remark To change the contents of CR5*n*, you must call TM5n_Stop before calling this API function.

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    TM5n_ChangeTimerCondition ( UCHAR value );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`UCHAR valueg;`	The value to set in CR5*n*

[Return value]

None.

TM5*n*_ChangeDuty

Changes the duty ratio of the PWM signal output to the TO5*n* pin.

Remark This API function can only be called when the 8-bit timer/event counter 5*n* is being used for PWM output.

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
void    TM5n_ChangeDuty ( UCHAR ratio );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	UCHAR *ratio*;	Duty ratio (0 to 100, unit: %)

Remark The value set to duty ratio *ratio* must be in base 10 notation.

[Return value]

None.

[Example]

The example below shows changing the duty ratio to 25%.

[CG_main.c]

```
#include    "CG_macrodriver.h"
void main ( void ) {
    UCHAR   ratio = 25;
    ......
    TM50_Start ();                  /* Start count */
    ......
    TM50_ChangeDuty ( ratio );      /* Change duty ratio */
    ......
}
```

TMH*n*_Init

Performs initialization necessary to control 8-bit timer H*n* functions.

[Classification]

CG_timer.c

[Syntax]

```
void    TMHn_Init ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMH*n*_UserInit

Performs user-defined initialization relating to the 8-bit timer H*n*.

Remark This API function is called as the TMHn_Init callback routine.

[Classification]

CG_timer_user.c

[Syntax]

```
void    TMHn_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMH*n*_Start

Starts the count for 8-bit timer H*n*.

Remark The time from the call to this API function to the start of counting depends on the type of the function in question (e.g. interval timer, square-wave output, or PWM output).

[Classification]

CG_timer.c

[Syntax]

```
void    TMHn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMH*n*_Stop

Ends the count for 8-bit timer H*n*.

[Classification]

CG_timer.c

[Syntax]

```
void    TMHn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMH*n*_ChangeTimerCondition

Changes the value of 8-bit timer H compare register 0*n*/1*n* (CMP0*n*/CMP1*n*).

Remark To change the contents of CMP0*n*/CMP1*n*, you must call TMHn_Stop before calling this API function.

[Classification]

CG_timer.c

[Syntax]

```
#include    "CG_macrodriver.h"
#include    "CG_timer.h"
MD_STATUS   TMHn_ChangeTimerCondition ( UCHAR *array_reg, UCHAR array_num );
```

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	`UCHAR *array_reg;`	Pointer to the area storing the value to set in the target register
I	`UCHAR array_num;`	The register to change 1: CMP0*n* 2: CMP0*n*, CMP1*n*

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

TMH*n*_ChangeDuty

Changes the duty ratio of the PWM signal output to the TOH*n* pin.

Remark This API function can only be called when the 8-bit timer H*n* is being used for PWM output.

[Classification]

CG_timer.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

I/O	Argument	Description
I	UCHAR *ratio*;	Duty ratio (0 to 100, unit: %)

Remark The value set to duty ratio *ratio* must be in base 10 notation.

[Return value]

None.

[Example]

The example below shows changing the duty ratio to 25%.

[CG_main.c]

```
#include    "CG_macrodriver.h"
void main ( void ) {
    UCHAR   ratio = 25;
    ......
    TMH0_Start ();                  /* Start count */
    ......
    TMH0_ChangeDuty ( ratio );      /* Change duty ratio */
    ......
}
```

TMH1_CarrierOutputEnable

Begins carrier pulse output of the 8-bit timer H1 (carrier generator mode).

[Classification]

CG_timer.c

[Syntax]

```
void    TMH1_CarrierOutputEnable ( void );
```

[Argument(s)]

None.

[Return value]

None.

TMH1_CarrierOutputDisable

Ends carrier pulse output of the 8-bit timer H1 (carrier generator mode).

[Classification]

CG_timer.c

[Syntax]

```
void    TMH1_CarrierOutputDisable ( void );
```

[Argument(s)]

None.

[Return value]

None.

C.3.9 Watchdog Timer

Below is a list of API functions output by Code Generator for watchdog timer use.

Table C-10. API Functions: [Watchdog Timer]

API Function Name	Function
WDT_Restart	Clears the watchdog timer counter and resumes counting.

WDT_Restart

Clears the watchdog timer counter and resumes counting.

[Classification]

CG_wdt.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

C.3.10 Real-time Clock

Below is a list of API functions output by Code Generator for real-time counter use.

Table C-11. API Functions: [Real-time Clock]

API Function Name	Function
RTC_Init	Performs initialization necessary to control real-time counter functions.
RTC_UserInit	Performs user-defined initialization relating to the real-time counter.
RTC_PowerOff	Halts the clock supplied to the real-time counter.
RTC_CounterEnable	Starts the count of the real-time counter (year, month, weekday, day, hour, minute, second).
RTC_CounterDisable	Ends the count of the real-time counter (year, month, weekday, day, hour, minute, second).
RTC_SetHourSystem	Sets the clock type (12-hour or 24-hour clock) of the real-time counter.
RTC_CounterSet	Sets the counter value (year, month, weekday, day, hour, minute, second) of the real-time counter.
RTC_CounterGet	Reads the counter value (year, month, weekday, day, hour, minute, second) of the real-time counter.
RTC_ConstPeriodInterruptEnable	Sets the cycle of the interrupts INTRTC, then starts the cyclic interrupt function.
RTC_ConstPeriodInterruptDisable	Ends the cyclic interrupt function.
RTC_ConstPeriodInterruptCallback	Performs processing in response to the cyclic interrupt INTRTC.
RTC_AlarmEnable	Starts the alarm interrupt function.
RTC_AlarmDisable	Ends the alarm interrupt function.
RTC_AlarmSet	Sets the alarm conditions (weekday, hour, minute).
RTC_AlarmGet	Reads the alarm conditions (weekday, hour, minute).
RTC_AlarmInterruptCallback	Performs processing in response to the alarm interrupt INTRTC.
RTC_IntervalStart	Starts the interval interrupt function.
RTC_IntervalStop	Ends the interval interrupt function.
RTC_IntervalInterruptEnable	Sets the cycle of the interrupts INTRTCI, then starts the interval interrupt function.
RTC_IntervalInterruptDisable	Ends the interval interrupt function.
RTC_RTC1HZ_OutputEnable	Enables output of the real-time counter correction clock (1 Hz) to the RTC1HZ pin.
RTC_RTC1HZ_OutputDisable	Disables output of the real-time counter correction clock (1 Hz) to the RTC1HZ pin.
RTC_RTCCL_OutputEnable	Enables output of the real-time counter clock (32 kHz source) to the RTCCL pin.
RTC_RTCCL_OutputDisable	Disables output of the real-time counter clock (32 kHz source) to the RTCCL pin.
RTC_RTCDIV_OutputEnable	Enables output of the real-time counter clock (32 kHz cycle) to the RTCDIV pin.
RTC_RTCDIV_OutputDisable	Disables output of the real-time counter clock (32 kHz cycle) to the RTCDIV pin.
RTC_ChangeCorrectionValue	Changes the timing and correction value for correcting clock errors.

RTC_Init

Performs initialization necessary to control real-time counter functions.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_UserInit

Performs user-defined initialization relating to the real-time counter.

Remark This API function is called as the RTC_Init callback routine.

[Classification]

CG_rtc_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_PowerOff

Halts the clock supplied to the real-time counter.

Remark Calling this API function changes the real-time counter to reset status. For this reason, writes to the control registers (e.g. real-time counter control register 0: RTCC0) after this API function is called are ignored.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_CounterEnable

Starts the count of the real-time counter (year, month, weekday, day, hour, minute, second).

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_CounterDisable

Ends the count of the real-time counter (year, month, weekday, day, hour, minute, second).

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_SetHourSystem

Sets the clock type (12-hour or 24-hour clock) of the real-time counter.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	`enum RTCHourSystem hoursystem;`	Clock type HOUR12: 12-hour clock HOUR24: 24-hour clock

[Return value]

Macro	Description
MD_OK	Normal completion
MD_BUSY1	Executing count process (before change to setting)
MD_BUSY2	Stopping count process (after change to setting)
MD_ARGERROR	Invalid argument specification

Remark If MD_BUSY1 or MD_BUSY2 is returned, it may be because the counter-operation is stopped, or the counter operation start wait time is too short, so make the value of the RTC_WAITTIME macro defined in the header file "CG_rtc.h" larger.

[Example]

Below is an example of setting the clock type to the 24-hour clock.

[CG_main.c]

```
#include    "CG_rtc.h"
void main ( void ) {
    ......
    RTC_CounterEnable ();           /* Start count */
    ......
    RTC_SetHourSystem ( HOUR24 );   /* Set clock type */
    ......
```

```
}
```

RTC_CounterSet

Sets the counter value (year, month, weekday, day, hour, minute, second) of the real-time counter.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	`struct RTCCounterValue counterwriteval;`	Counter value

Remark Below is an example of the structure RTCCounterValue (counter value) for the real-time counter.

```
struct  RTCCounterValue {
    UCHAR   Sec;    /* second */
    UCHAR   Min;    /* Minute */
    UCHAR   Hour;   /* Hour */
    UCHAR   Day;    /* Day */
    UCHAR   Week;   /* Weekday (0: Sunday, 6: Saturday) */
    UCHAR   Month;  /* Month */
    UCHAR   Year;   /* Year */
};
```

[Return value]

Macro	Description
MD_OK	Normal completion
MD_BUSY1	Executing count process (before change to setting)
MD_BUSY2	Stopping count process (after change to setting)

Remark If MD_BUSY1 or MD_BUSY2 is returned, it may be because the counter-operation is stopped, or the counter operation start wait time is too short, so make the value of the RTC_WAITTIME macro defined in the header file "CG_rtc.h" larger.

[Example]

The example below shows the counter value of the real-time counter being set to "2008/12/25 (Thu.) 17:30:00".

[CG_main.c]

```
#include    "CG_rtc.h"
void main ( main ) {
    struct  RTCCounterValue counterwriteval;
    ......
    RTC_CounterEnable ();               /* Start count */
    ......
    counterwriteval.Year = 0x08;
    counterwriteval.Month = 0x12;
    counterwriteval.Day = 0x25;
    counterwriteval.Week = 0x05;
    counterwriteval.Hour = 0x17;
    counterwriteval.Min = 0x30;
    counterwriteval.Sec = 0;
    RTC_SetHourSystem ( HOUR24 );       /* Set clock type */
    RTC_CounterSet ( counterwriteval ); /* Set counter value */
    ......
}
```

RTC_CounterGet

Reads the counter value (year, month, weekday, day, hour, minute, second) of the real-time counter.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
O	`struct RTCCounterValue *counterreadval;`	Pointer to structure in which to store the counter value being read

Remark See RTC_CounterSet for details about the RTCCounterValue counter value.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_BUSY1	Executing count process (before reading)
MD_BUSY2	Stopping count process (after reading)

Remark If MD_BUSY1 or MD_BUSY2 is returned, it may be because the counter-operation is stopped, or the counter operation start wait time is too short, so make the value of the RTC_WAITTIME macro defined in the header file "CG_rtc.h" larger.

[Example]

Below is an example of reading the counter value of the real-time counter.

[CG_main.c]

```
#include    "CG_rtc.h"
void main ( void ) {
    struct  RTCCounterValue counterreadval;
    ......
    RTC_CounterEnable ();               /* Start count */
    ......
    RTC_CounterGet ( &counterreadval ); /* Read count value */
    ......
```

```
}
```

RTC_ConstPeriodInterruptEnable

Sets the cycle of the interrupts INTRTC, then starts the cyclic interrupt function.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	`enum RTCINTPeriod period;`	Interrupt INTRTC cycle HALFSEC: 0.5 seconds ONESEC: 1 second ONEMIN: 1 minute ONEHOUR: 1 hour ONEDAY: 1 day ONEMONTH: 1 month

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

[Example]

Below is an example of setting the cycle of the interrupts INTRTC, then starting the cyclic interrupt function.

[CG_main.c]

```
#include    "CG_rtc.h"
void main ( void ) {
    ......
    RTC_ConstPeriodInterruptDisable ();         /* End of cyclic interrupt function */
    ......
    RTC_ConstPeriodInterruptEnable ( HALFSEC ); /* Start of cyclic interrupt function */
    ......
}
```

RTC_ConstPeriodInterruptDisable

Ends the cyclic interrupt function.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_ConstPeriodInterruptCallback

Performs processing in response to the cyclic interrupt INTRTC.

Remark This API function is called as the callback routine of interrupt process MD_INTRTC corresponding to the cyclic interrupt INTRTC.

[Classification]

CG_rtc_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_AlarmEnable

Starts the alarm interrupt function.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_AlarmDisable

Ends the alarm interrupt function.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_AlarmSet

Sets the alarm conditions (weekday, hour, minute).

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

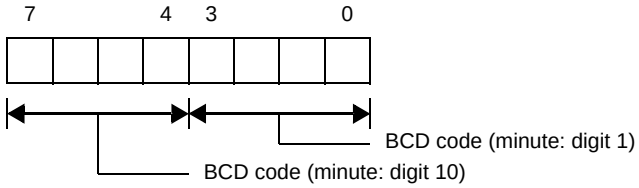
I/O	Argument	Description
I	`struct RTCAlarmValue alarmval;`	Alarm conditions (weekday, hour, minute)

Remark Below is shown the structure RTCAlarmValue (alarm conditions).

```
struct  RTCAlarmValue {
    UCHAR   Alarmwm;    /* Minute */
    UCHAR   Alarmwh;    /* Hour */
    UCHAR   Alarmww;    /* Weekday */
};
```

- Alarmwm (Minute)

  Below are shown the meanings of each bit of the structure member Alarmwm.

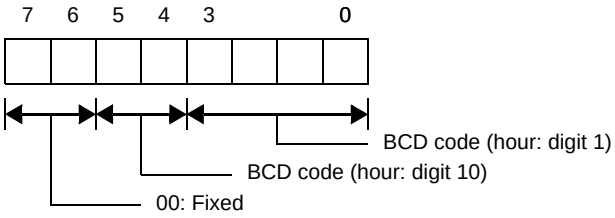


- Alarmwh (Hour)

  Below are shown the meanings of each bit of the structure member Alarmwh. If the real-time counter is set to the 12-hour clock, then bit 5 has the following meaning.

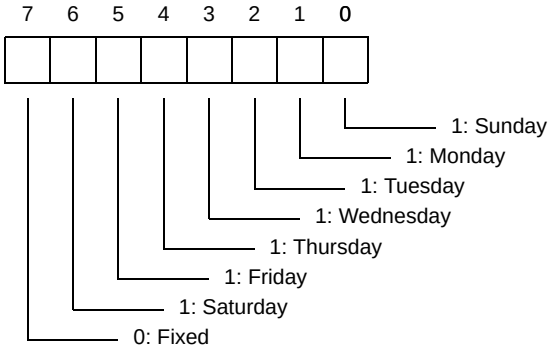
  0: AM

  1: PM



- Alarmww (Weekday)

  Below are shown the meanings of each bit of the structure member Alarmww.



[Return value]

None.

[Example 1]

The example below shows the alarm conditions being set to "Monday/Tuesday/Wednesday at 17:30".

[CG_main.c]

```
#include    "CG_rtc.h"
void main ( void ) {
    struct  RTCAlarmValue   alarmval;
    ......
    RTC_AlarmEnable ();             /* Start alarm interrupt function */
    RTC_CounterEnable ();           /* Start count */
    ......
    RTC_SetHourSystem ( HOUR24 );   /* Set clock type */
    alarmval.Alarmww = 0xe;
    alarmval.Alarmwh = 0x17;
    alarmval.Alarmwm = 0x30;
    RTC_AlarmSet ( alarmval );      /* Set conditions */
    ......
}
```

[Example 2]

The example below shows the alarm conditions being set to "Saturday/Sunday (time left unchanged)".

[CG_main.c]

```
#include    "CG_rtc.h"
void main ( void ) {
    struct  RTCAlarmValue   alarmval;
    ......
    RTC_AlarmEnable ();             /* Start alarm interrupt function */
    ......
    RTC_AlarmSet ( alarmval );      /* Change conditions */
    ......
}
```

RTC_AlarmGet

Reads the alarm conditions (weekday, hour, minute).

[Classification]

CG_rtc.c

[Syntax]

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

Remark See RTC_AlarmSet for details about RTCAlarmValue (alarm conditions).

[Argument(s)]

I/O	Argument	Description
O	`struct RTCAlarmValue *alarmval;`	Pointer to structure in which to store the conditions being read

[Return value]

None.

[Example]

The example below shows the alarm conditions being read.

[CG_main.c]

```
#include    "CG_rtc.h"
void main ( void ) {
    struct  RTCAlarmValue   alarmval;
    ......
    RTC_AlarmEnable ();         /* Start alarm interrupt function */
    ......
    RTC_AlarmGet ( &alarmval ); /* Read conditions */
    ......
}
```

RTC_AlarmInterruptCallback

Performs processing in response to the alarm interrupt INTRTC.

Remark This API function is called as the callback routine of interrupt process MD_INTRTC corresponding to the alarm interrupt INTRTC.

[Classification]

CG_rtc_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_IntervalStart

Starts the interval interrupt function.

Remark After setting the cycle of the interrupts INTRTCI, call RTC_IntervalInterruptEnable to start the interval interrupt function.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_IntervalStop

Ends the interval interrupt function.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_IntervalInterruptEnable

Sets the cycle of the interrupts INTRTCI, then starts the interval interrupt function.

Remark Call RTC_IntervalStart to start the interval interrupt function without setting the cycle of the interrupts INTRTCI.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	`enum RTCINTInterval interval;`	Interrupt INTRTCI cycle INTERVAL0: 2^6/fSUB INTERVAL1: 2^7/fSUB INTERVAL2: 2^8/fSUB INTERVAL3: 2^9/fSUB INTERVAL4: 2^10/fSUB INTERVAL5: 2^11/fSUB INTERVAL6: 2^12/fSUB

Remark fSUB is the frequency of the subsystem clock.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

[Example]

Below is an example of changing the interval, the restarting the interval interrupt function.

[CG_main.c]

```
#include    "CG_rtc.h"
void main ( void ) {
    ......
    RTC_IntervalStart ();                    /* Start interval interrupt function */
    ......
```

```
    RTC_IntervalStop ();                        /* End interval interrupt function */
    ......
    RTC_IntervalInterruptEnable ( INTERVAL6 );  /* Start interval interrupt function */
    ......
}
```

RTC_IntervalInterruptDisable

Ends the interval interrupt function.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_RTC1HZ_OutputEnable

Enables output of the real-time counter correction clock (1 Hz) to the RTC1HZ pin.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_RTC1HZ_OutputDisable

Disables output of the real-time counter correction clock (1 Hz) to the RTC1HZ pin.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_RTCCL_OutputEnable

Enables output of the real-time counter clock (32 kHz source) to the RTCCL pin.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_RTCCL_OutputDisable

Disables output of the real-time counter clock (32 kHz source) to the RTCCL pin.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_RTCDIV_OutputEnable

Enables output of the real-time counter clock (32 kHz cycle) to the RTCDIV pin.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_RTCDIV_OutputDisable

Disables output of the real-time counter clock (32 kHz cycle) to the RTCDIV pin.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

RTC_ChangeCorrectionValue

Changes the timing and correction value for correcting clock errors.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	`enum RTCCorectionTiming timing;`	When clock errors are corrected EVERY20S: When the seconds digits are 00, 20 or 40 EVERY60S: When the seconds digits are 00
I	`UCHAR corectval;`	Clock error correction value

Remark This API function does not correct clock errors if correction value *corectVal* is set to 0x0, 0x1, 0x40 or 0x41.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

C.3.11 Clock Output

Below is a list of API functions output by Code Generator for clock output use.

Table C-12. API Functions: [Clock Output]

API Function Name	Function
PCL_Init	Performs initialization necessary to control clock output control circuit functions.
PCL_UserInit	Performs user-defined initialization relating to the clock output control circuits.
PCL_Start	Starts clock output.
PCL_Stop	Ends clock output.
PCL_ChangeFreq	Changes the output clock to the PCL pin.

PCL_Init

Performs initialization necessary to control clock output control circuit functions.

[Classification]

CG_pcl.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

PCL_UserInit

Performs user-defined initialization relating to the clock output control circuits.

Remark This API function is called as the PCL_Init callback routine.

[Classification]

CG_pcl_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

PCL_Start

Starts clock output.

[Classification]

CG_pcl.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

PCL_Stop

Ends clock output.

[Classification]

CG_pcl.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

PCL_ChangeFreq

Changes the output clock to the PCL pin.

Remark The value specified in parameter *clock* is set to clock output select register (CKS).

[Classification]

CG_pcl.c

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	`enum PCLclock clock;`	Output clock type FPRS: fPRS FPRS2: fPRS/2 FPRS4: fPRS/4 FPRS8: fPRS/8 FPRS16: fPRS/16 FPRS32: fPRS/2048 FPRS64: fPRS/4096 FPRS128: fPRS/8192 SUBCLOCK: fSUB

Remark fPRS is the main system clock frequency; fSUB is the subsystem clock frequency.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ARGERROR	Invalid argument specification

C.3.12 LVI

Below is a list of API functions output by Code Generator for low-voltage detector use.

Table C-13. API Functions: [LVI]

API Function Name	Function
LVI_Init	Performs initialization necessary to control low-voltage detector functions.
LVI_UserInit	Performs user-defined initialization relating to the low-voltage detector.
LVI_InterruptModeStart	Starts low-voltage detection (when in interrupt generation mode).
LVI_ResetModeStart	Starts low-voltage detection (when in internal reset mode).
LVI_Stop	Stops low-voltage detection.
LVI_SetLVILevel	Sets the low-voltage detection level.

LVI_Init

Performs initialization necessary to control low-voltage detector functions.

[Classification]

CG_lvi.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

LVI_UserInit

Performs user-defined initialization relating to the low-voltage detector.

Remark This API function is called as the LVI_Init callback routine.

[Classification]

CG_lvi_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

LVI_InterruptModeStart

Starts low-voltage detection (when in interrupt generation mode).

[Classification]

CG_lvi.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

[Example]

The example below shows the detection of low voltage when the operation mode is interrupt generation mode (generate the interrupt INTLVI).

[CG_main.c]

```
void main ( void ) {
    ......
    LVI_InterruptModeStart ( );         /* Start low-voltage detection */
    ......
}
```

[CG_lvi_user.c]

```
__interrupt void MD_INTLVI ( void ) {   /* Interrupt processing for INTLVI */
    if ( LVIF == 1 ) {                  /* Trigger identification: Check LVIF flag */
        ...... /* Handle case when "power voltage (VDD) < detected voltage (VLVI)" detected */
    } else {
        ...... /* Handle case when "power voltage (VDD) >= detected voltage (VLVI)" detected */
    }
}
```

LVI_ResetModeStart

Starts low-voltage detection (when in internal reset mode).

[Classification]

CG_lvi.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR	Exit with error (abend) - The object of low voltage detection is external voltage (V_{DD}), and power voltage (V_{DD}) <= detected voltage (V_{LVI}). - The object of low voltage detection is external input voltage (EXLVI), and external input voltage (EXLVI) <= detected voltage (V_{EXLVI}).

LVI_Stop

Stops low-voltage detection.

[Classification]

CG_lvi.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

LVI_SetLVILevel

Sets the low-voltage detection level.

Remarks 1. To change the low-voltage detection level, you must call LVI_Stop before calling this API function.

2. The value specified in parameter *level* is set to low-voltage detection level select register (LVIS).

[Classification]

CG_lvi.c

[Syntax]

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

[Argument(s)]

I/O	Argument	Description
I	`enum LVILevel level;`	Voltage level to detect as low voltage 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

Remark LVILEVEL10 to LVILEVEL15 can only be specified when the target device is a 78K0/Kx2-L.

[Return value]

Macro	Description
MD_OK	Normal completion
MD_ERROR	Exit with error (abend) - The target of low-voltage detection is external input voltage (EXLVI) from the external input pin.
MD_ARGERROR	Invalid argument specification

Remark The value MD_ERROR will only be returned when the target device is a 78K0/KB2-L or 78K0/KC2-L.

APPENDIX D INDEX

S

T

U

W

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Sep 01, 2012	-	First Edition issued

CubeSuite+ V1.03.00 User's Manual:
78K0 Design

Publication Date: Rev.1.00 Sep 01, 2012

Published by: Renesas Electronics Corporation

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

© 2012 Renesas Electronics Corporation. All rights reserved.

Colophon 1.3

CubeSuite+ V1.03.00

Renesas Electronics Corporation

R20UT2138EJ0100