

CubeSuite+ V1.03.00

Integrated Development Environment

User's Manual: V850 Design

Target Device

V850 Family

All information contained in these materials, 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>).

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.

How to Use This Manual

This manual describes the role of the CubeSuite+ integrated development environment for developing applications and systems for V850 family, and provides an outline of its features.

CubeSuite+ is an integrated development environment (IDE) for V850 family, 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.

CHAPTER 1 GENERAL
CHAPTER 2 FUNCTIONS (Pin Configurator)
CHAPTER 3 FUNCTIONS (Code Generator)
APPENDIX A WINDOW REFERENCE
APPENDIX B OUTPUT FILES
APPENDIX C API FUNCTIONS
APPENDIX D INDEX

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: | <u>Higher</u> digits on the left and lower digits on the right |
| Active low representation: | 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 | R2OUT2133E |
| | V850 Design | This manual |
| | R8C Design | R2OUT2135E |
| | RL78 Design | R2OUT2136E |
| | 78K0R Design | R2OUT2137E |
| | 78K0 Design | R2OUT2138E |
| | RX Coding | R2OUT0767E |
| | V850 Coding | R2OUT0553E |
| | Coding for CX Compiler | R2OUT2139E |
| | R8C Coding | R2OUT0576E |
| | RL78, 78K0R Coding | R2OUT2140E |
| | 78K0 Coding | R2OUT2141E |
| | RX Build | R2OUT0768E |
| | V850 Build | R2OUT0557E |
| | Build for CX Compiler | R2OUT2142E |
| | R8C Build | R2OUT0575E |
| | RL78, 78K0R Build | R2OUT2143E |
| | 78K0 Build | R2OUT0783E |
| | RX Debug | R2OUT2175E |
| | V850 Debug | R2OUT2144E |
| | R8C Debug | R2OUT0770E |
| | RL78 Debug | R2OUT2145E |
| 78K0R Debug | R2OUT0732E | |
| 78K0 Debug | R2OUT0731E | |
| Analysis | R2OUT2146E | |
| Message | R2OUT2147E | |

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

TABLE OF CONTENTS

CHAPTER 1 GENERAL ... 7

- 1.1 Overview ... 7
- 1.2 Features ... 7

CHAPTER 2 FUNCTIONS (Pin Configurator) ... 8

- 2.1 Overview ... 8
- 2.2 Open Device Pin List Panel ... 10
 - 2.2.1 Select item ... 11
 - 2.2.2 Change display order ... 12
 - 2.2.3 Add column ... 13
 - 2.2.4 Delete column ... 13
- 2.3 Open Device Top View Panel ... 14
 - 2.3.1 Select shape of microcontroller ... 15
 - 2.3.2 Select color ... 16
 - 2.3.3 Select popup information ... 17
 - 2.3.4 Select additional information ... 18
- 2.4 Enter Information ... 19
- 2.5 Output Report Files ... 20
 - 2.5.1 Output device pin list ... 20
 - 2.5.2 Output device top view ... 21

CHAPTER 3 FUNCTIONS (Code Generator) ... 22

- 3.1 Overview ... 22
- 3.2 Open Code Generator Panel ... 23
- 3.3 Enter Information ... 24
 - 3.3.1 Input rule ... 24
 - 3.3.2 Icon indicating incorrect entry ... 25
 - 3.3.3 Icon indicating pin conflict ... 26
- 3.4 Confirm Source Code ... 27
- 3.5 Output Source Code ... 28
 - 3.5.1 Setting that determines whether or not to generate source code ... 29
 - 3.5.2 Change file name ... 30
 - 3.5.3 Change API function name ... 31
 - 3.5.4 Change output mode ... 32
 - 3.5.5 Change output destination folder ... 33
- 3.6 Output Report Files ... 34
 - 3.6.1 Change output format ... 36
 - 3.6.2 Change output destination ... 37

APPENDIX A WINDOW REFERENCE ... 38

A.1 Description ... 38

APPENDIX B OUTPUT FILES ... 84

B.1 Overview ... 84

B.2 Output File ... 84

APPENDIX C API FUNCTIONS ... 91

C.1 Overview ... 91

C.2 Output Function ... 91

C.3 Function Reference ... 100

C.3.1 System ... 102

C.3.2 External Bus ... 116

C.3.3 Port ... 119

C.3.4 Interrupt ... 125

C.3.5 Serial ... 136

C.3.6 A/D Converter ... 216

C.3.7 D/A Converter ... 225

C.3.8 Timer ... 231

C.3.9 Watch Timer ... 291

C.3.10 Real-time Clock ... 296

C.3.11 Real-Time Output ... 333

C.3.12 DMA ... 347

C.3.13 LVI ... 355

APPENDIX D INDEX ... 362

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 for microcontrollers manufactured by Renesas Electronics.

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 generators, ports, 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 the 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 the 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 the 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 the Pin area of the [Device Top View panel](#).

(5) Enter Information

Allows you to 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 the 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.

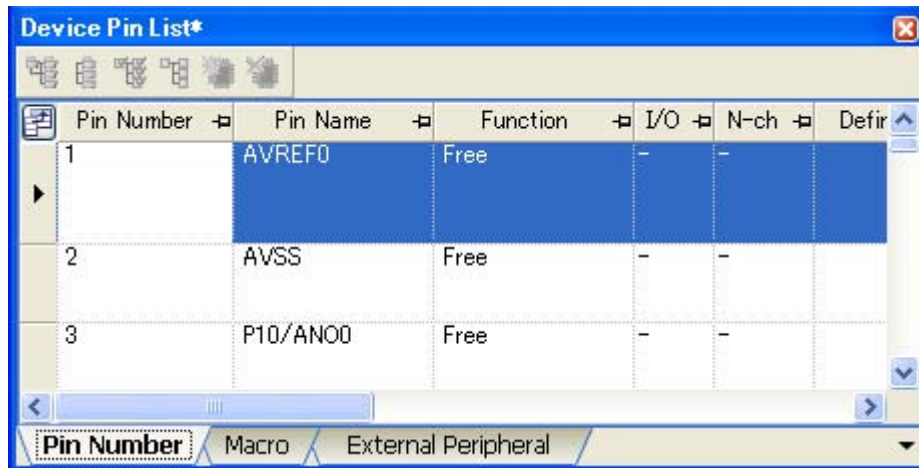
Remark See "CubeSuite+ Integrated Development Environment User's Manual: Start" for details on "Save project".

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 the 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.


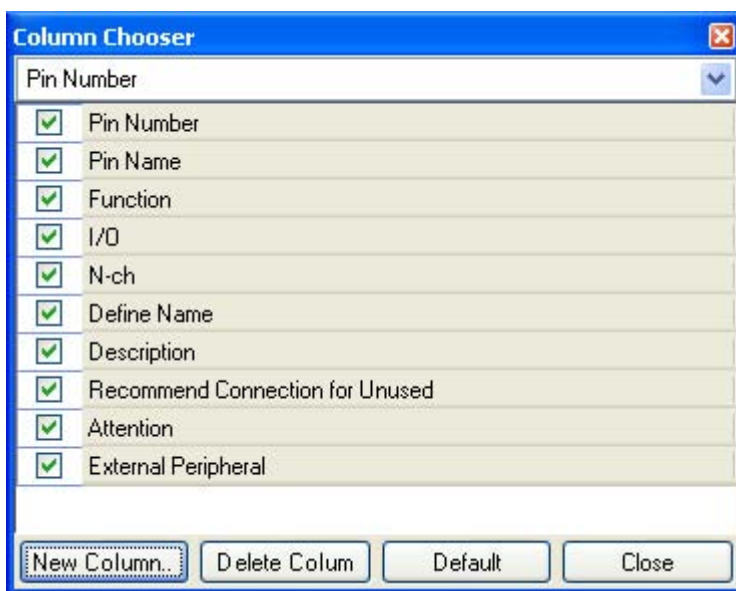
To select the item to be displayed, use the [Column Chooser dialog box](#) that opens by pressing the  button in the upper left corner of the device pin list.

Figure 2-2. Select Item



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

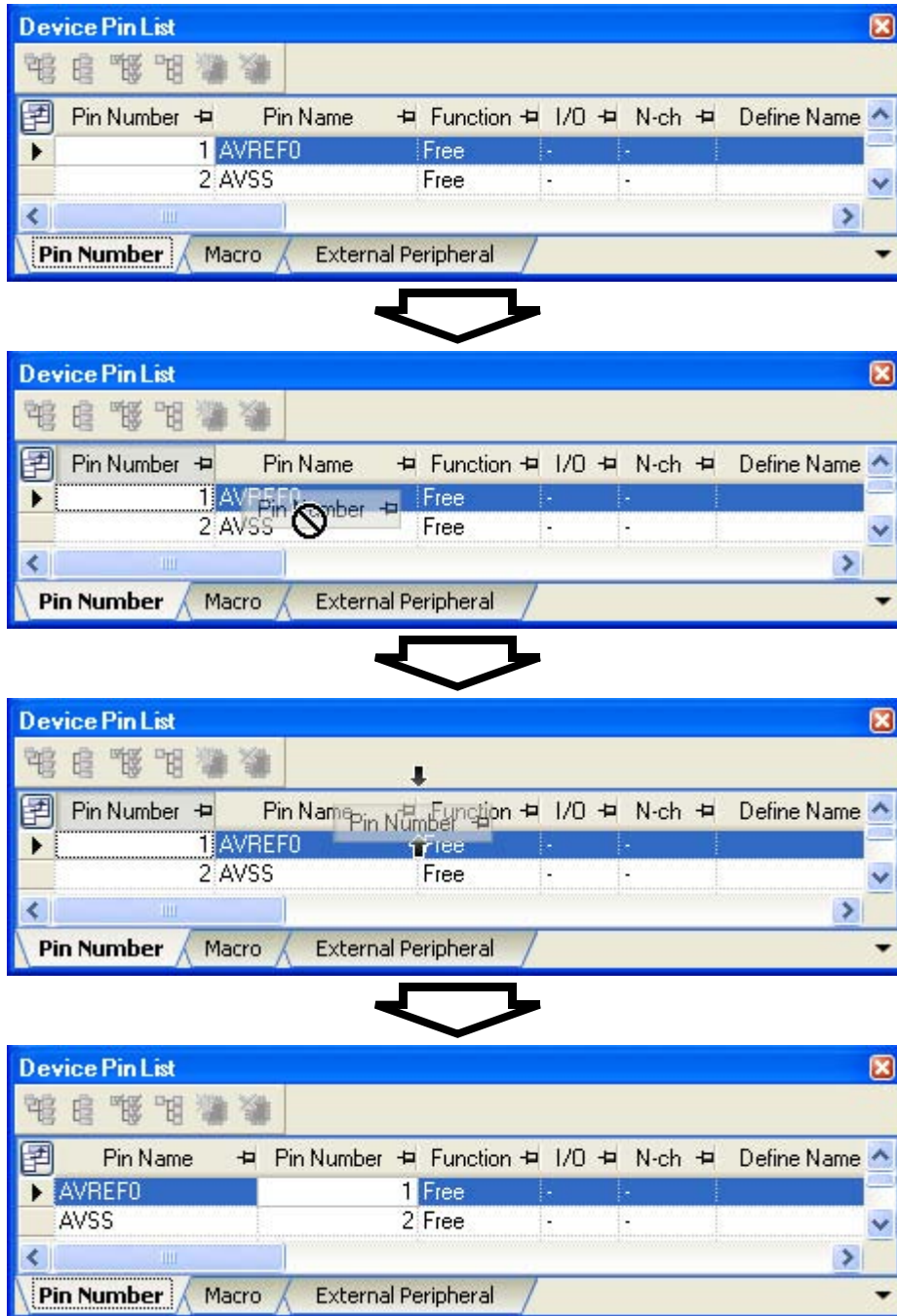
Table 2-1. Select Item


| | |
|-------------|--|
| 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 the 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



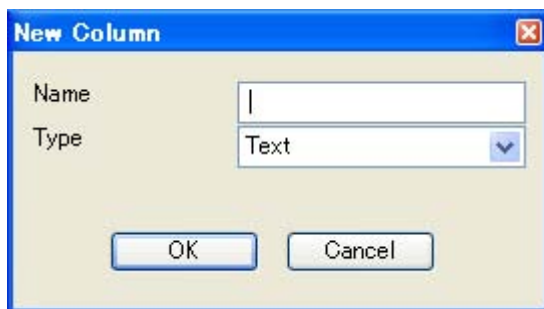
Remark To change the display order, click the  button in the upper left of the device pin list. The [Column Chooser dialog box](#) opens. Drag an item displayed in the dialog's select Items to display area, and drop it to the desired destination in the device pin list. This will change the display order.

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



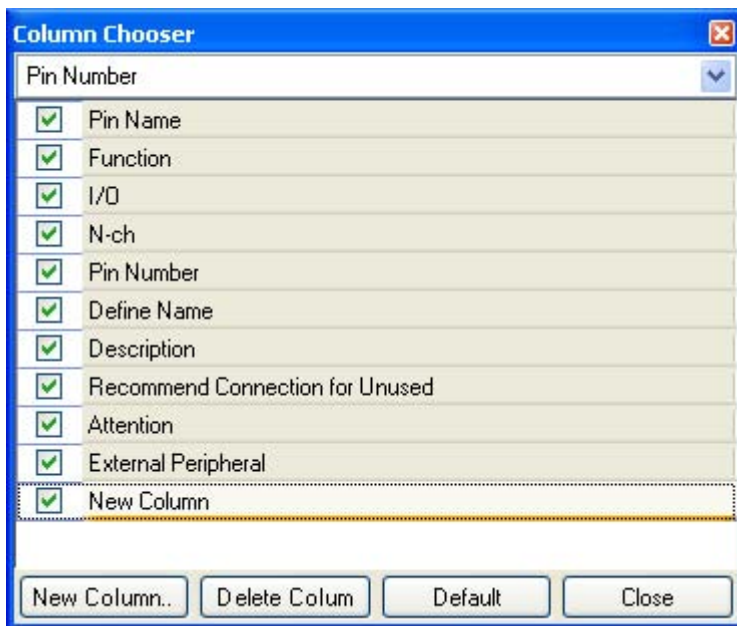
Remark On the device pin list, adding columns to the first level of [Macro] tab, [External Peripheral] tab is restricted.

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



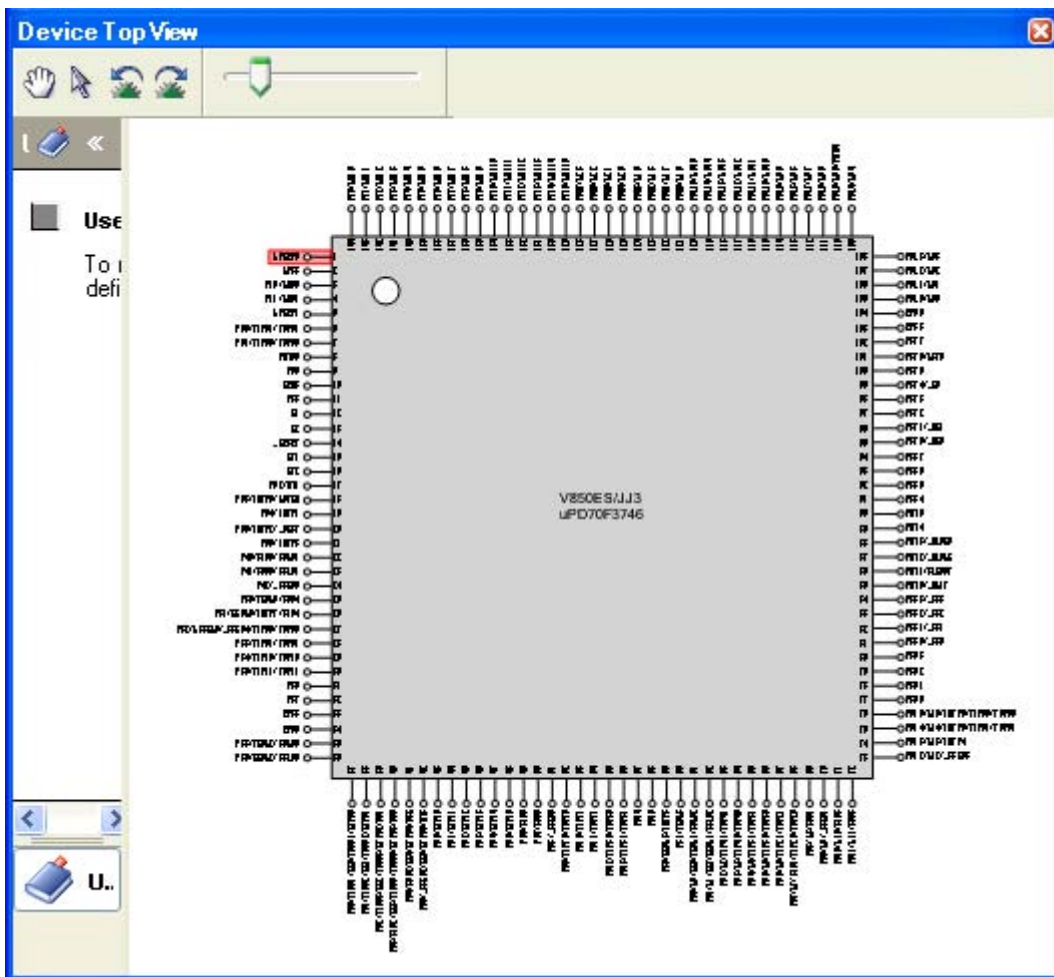
Remark You can only delete the column which you added using the New Column dialog box.

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



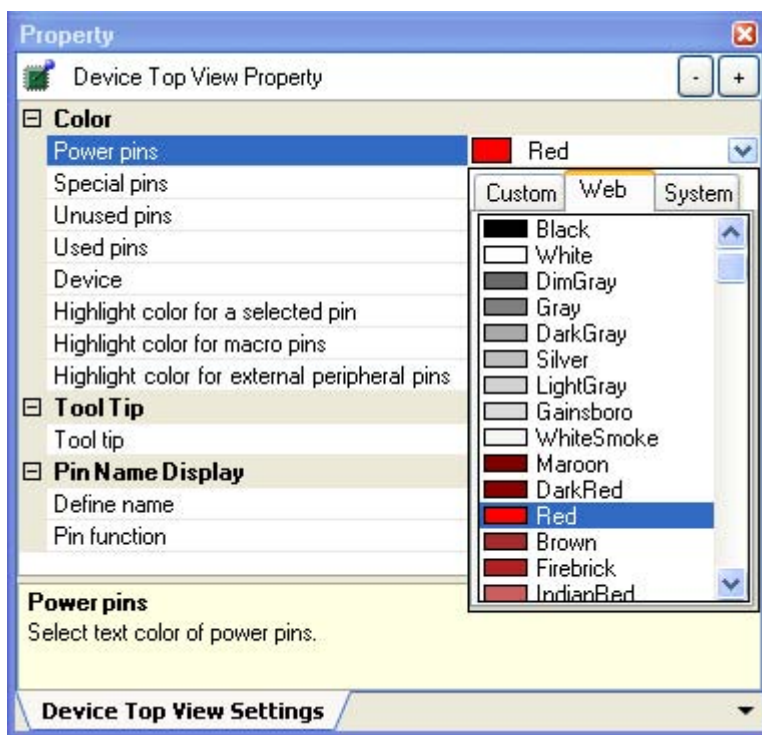
Remark Selection of the shape of the microcontroller is made using the order name (such as GC and GF).

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

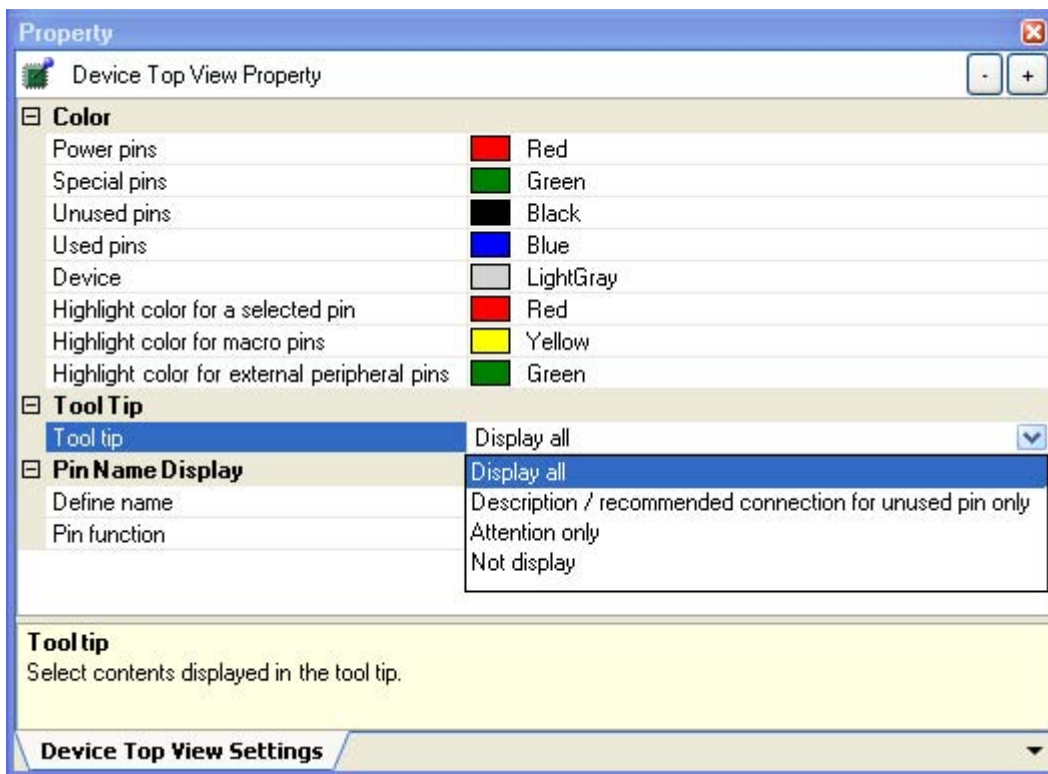
| Item | Outline |
|--|--|
| Power pins | Selects the display color for power pins (pins whose use is limited to power). |
| Special pins | Selects the display color for special pins (pins with specified uses). |
| Unused pins | Selects the display color for unused pins (dual-use pins with no use set 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

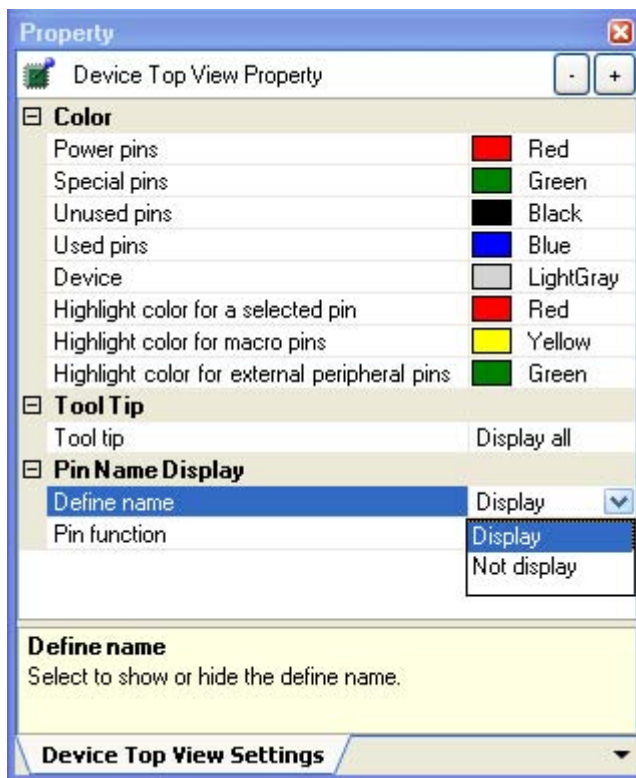
| Popup Information | Outline |
|--|---|
| Display all | Displays the "Description", "Recommend Connection for Unused", and "Attention" strings for the device pin list. |
| Description / recommended connection for unused pin only | Displays the "Description", and "Recommend Connection for Unused" strings 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 the 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).

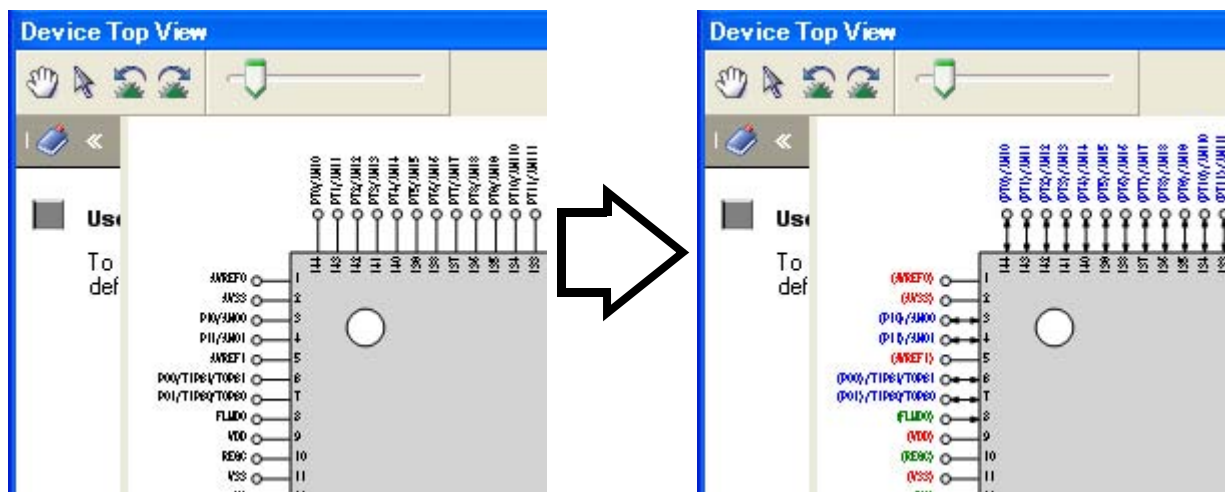
| | |
|------------------------|--|
| Display all | Displays functions selected via the device pin list's "Function" feature in parentheses. |
| Selected function only | Only display functions selected via the device pin list's "Function" feature in the device top view. |

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

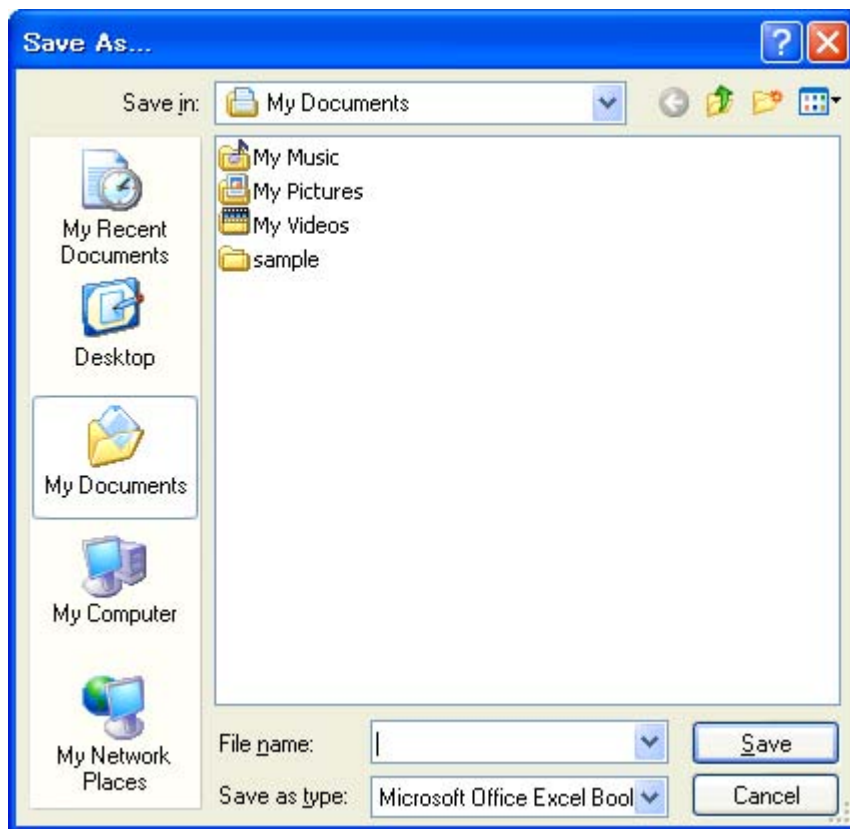
Output report files (files containing information configured using the Pin Configurator: device pin list and device top view) to the specified folder.

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 the 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



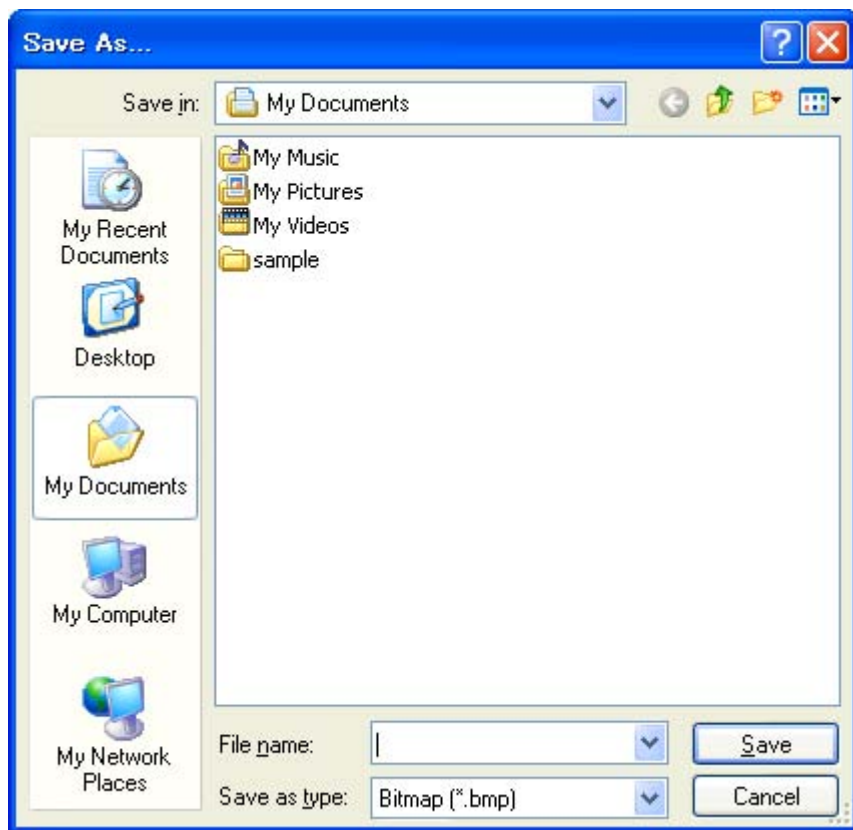
- Remarks 1.** If a device pin list has been already output, that list will be overwritten by selecting [File] menu >> [Save Pin List].
- 2.** The output format for the device pin list is limited to Microsoft Office Excel Book.

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 the 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 the CubeSuite+ panels that is needed to control peripheral functions provided by the microcontroller (clock generators, ports, etc.).

The following sections describe the operation procedures for the 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 generators, ports, etc.).

(4) Enter Information

Allows you to configure the information necessary to control the peripheral functions in the [Code Generator panel](#).

(5) Confirm Source Code

Allows you to 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 the Code Generator and a file containing information regarding the source code) to the specified folder.

(8) Save project

Save a project.

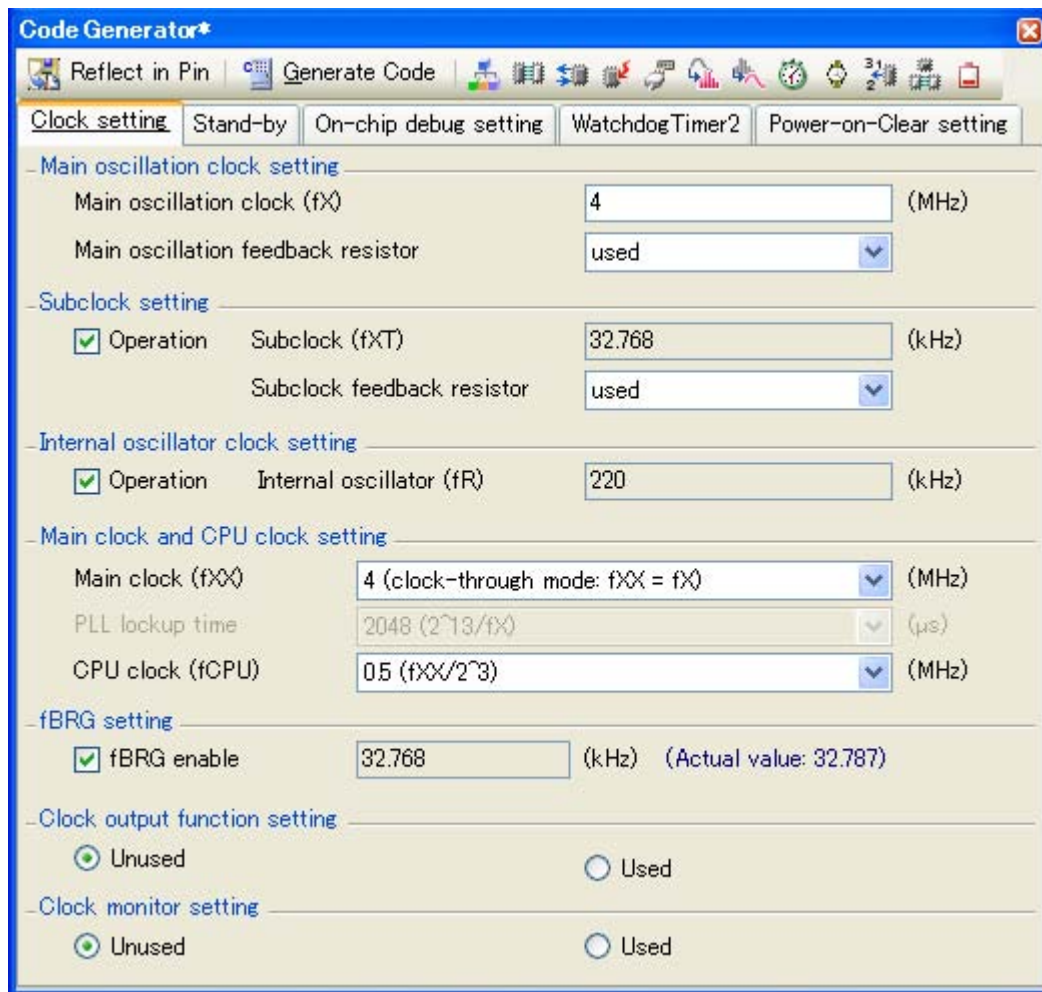
Remark See "CubeSuite+ Integrated Development Environment User's Manual: Start" for details on "Save project".

3.2 Open Code Generator Panel

Open the [Code Generator panel](#) to configure the information necessary to control the peripheral functions (clock generators, ports, 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 the 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.


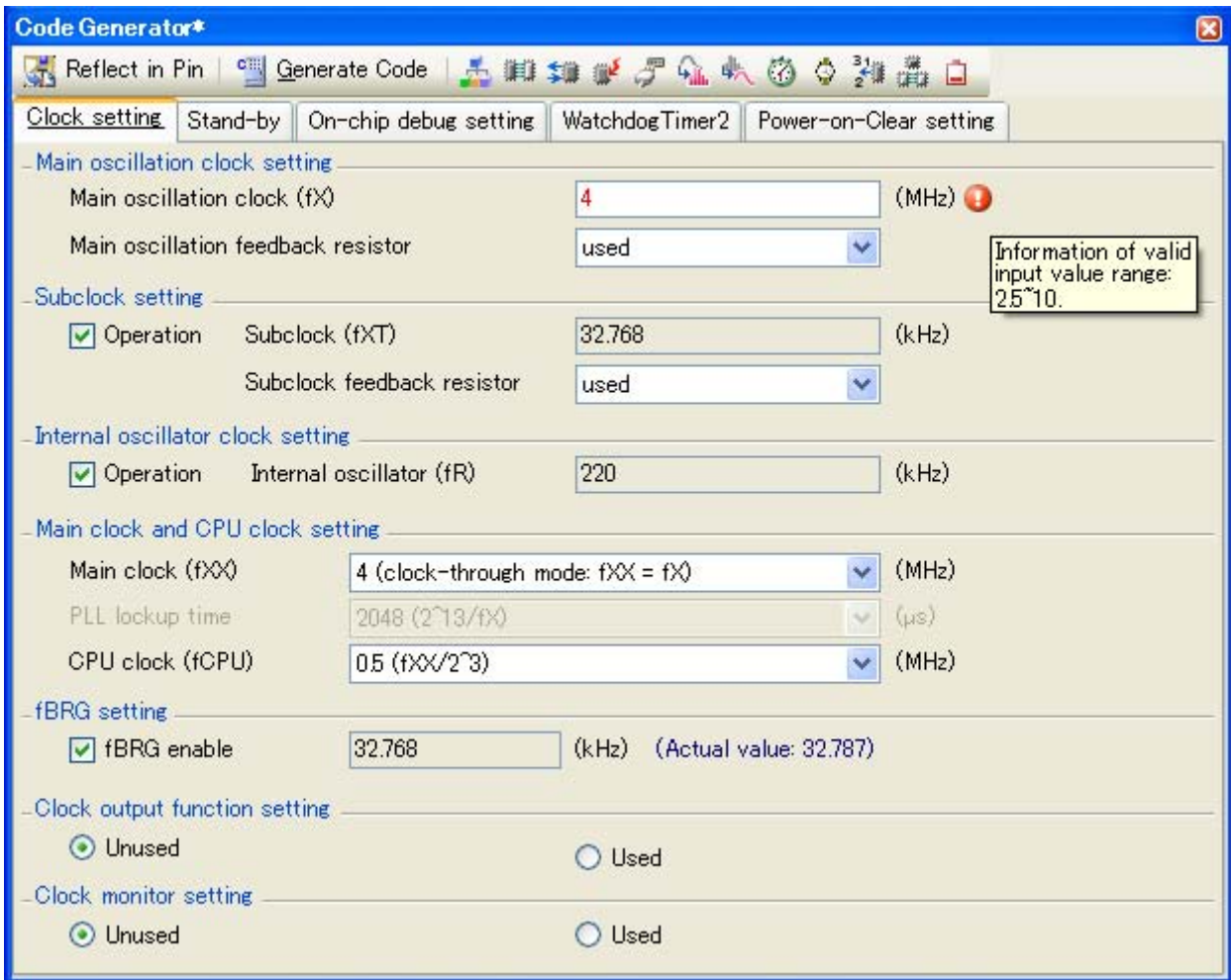

Remark If the mouse cursor is moved over the  icon, information regarding the string that should be entered (tips for correcting the entry) pops up.

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.


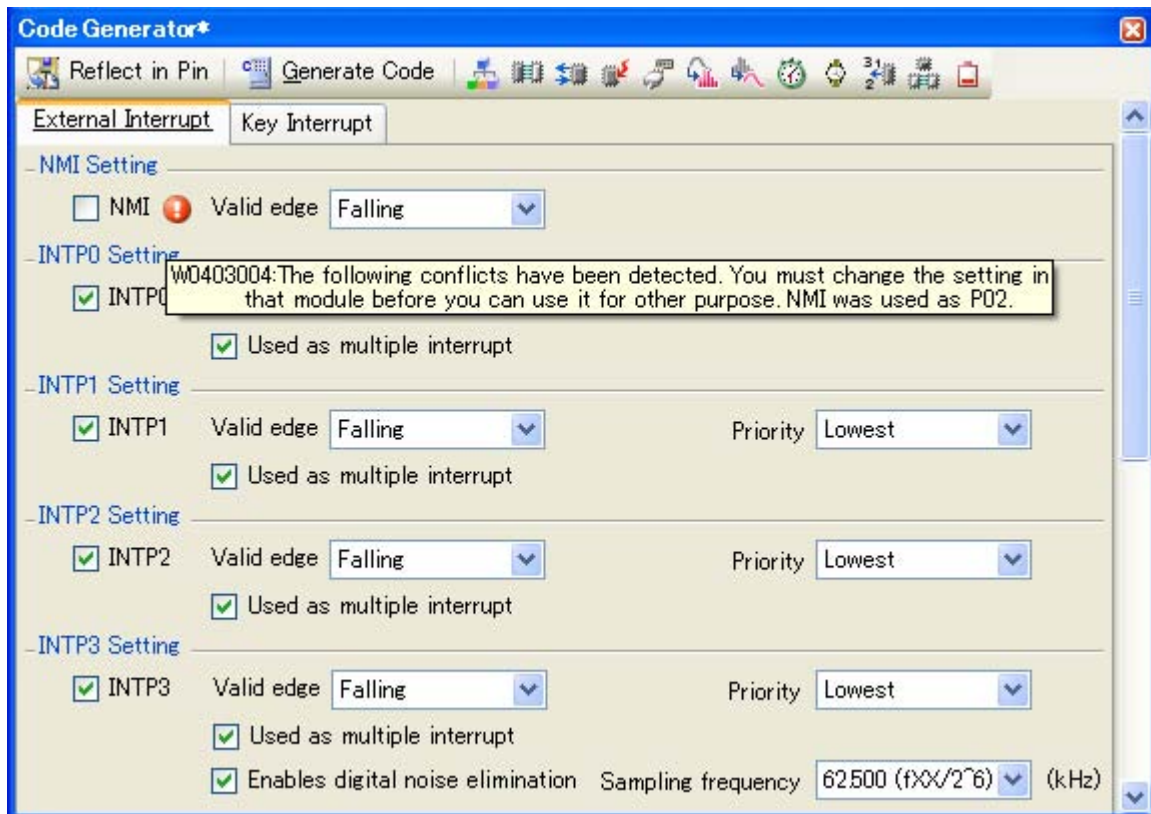
Remark If the mouse cursor is moved over the  icon, information regarding the conflict between the pins (tips for avoiding the conflict) pops up.

Figure 3-3. Icon Indicating Pin Conflict

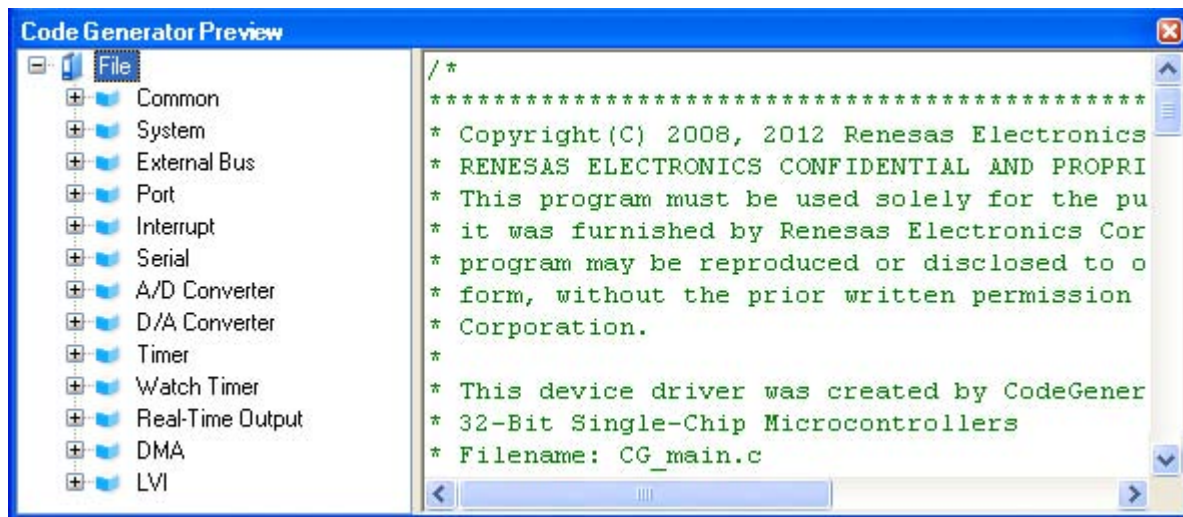


3.4 Confirm Source Code

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].

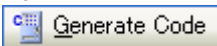
Figure 3-4. Confirm Source Code



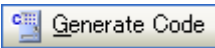
- Remarks 1. You can change the source code to be displayed by selecting the source file name or API function name in the [Code Generator Preview panel](#).
- 2. The following table displays the meaning of the color of the source code text displayed in the [Code Generator Preview panel](#).

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 |

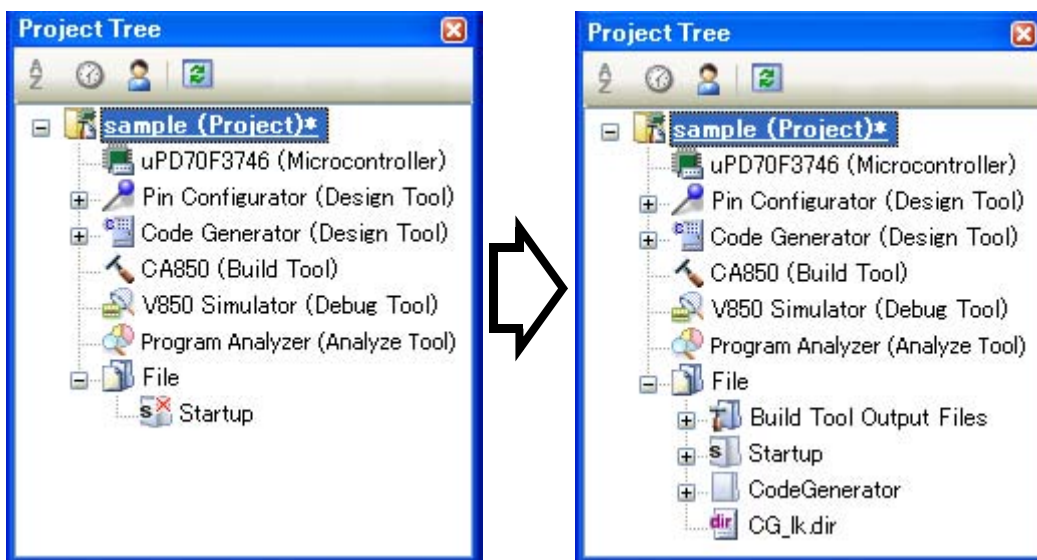
- 3. You cannot edit the source code within the [Code Generator Preview panel](#).
- 4. For some of the API functions (such as API functions for serial array units), values such as the SFR register value are calculated and finalized when the source code is generated (when the  button on the [Code Generator 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  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




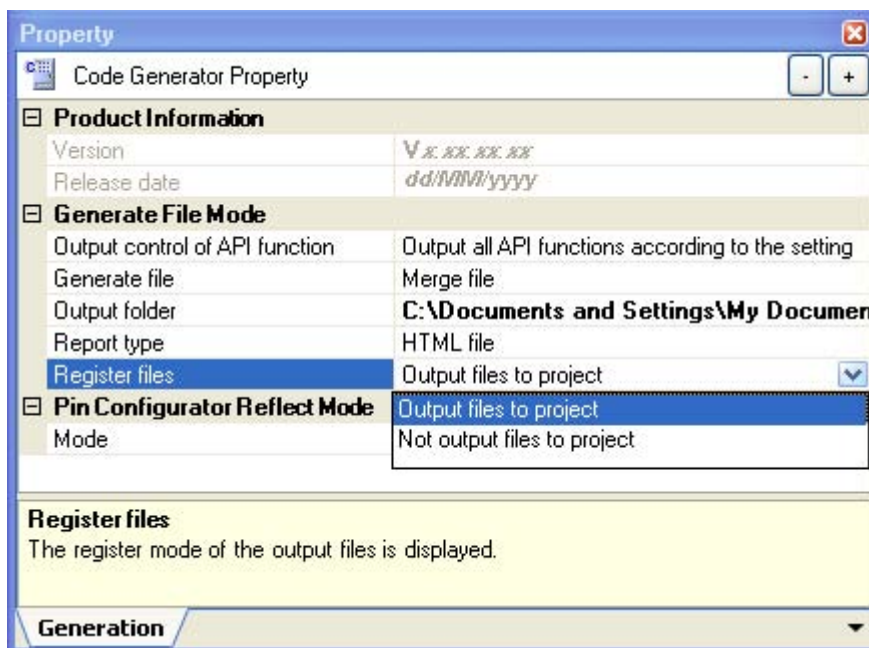
Remark In order to both output source files and add them to the project (display the corresponding source file names in the [Project Tree panel](#)) when you click the  button, you must open the [Property panel](#), and under [\[Generation\] tab](#) >> [\[Register files\]](#), specify "Output files to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

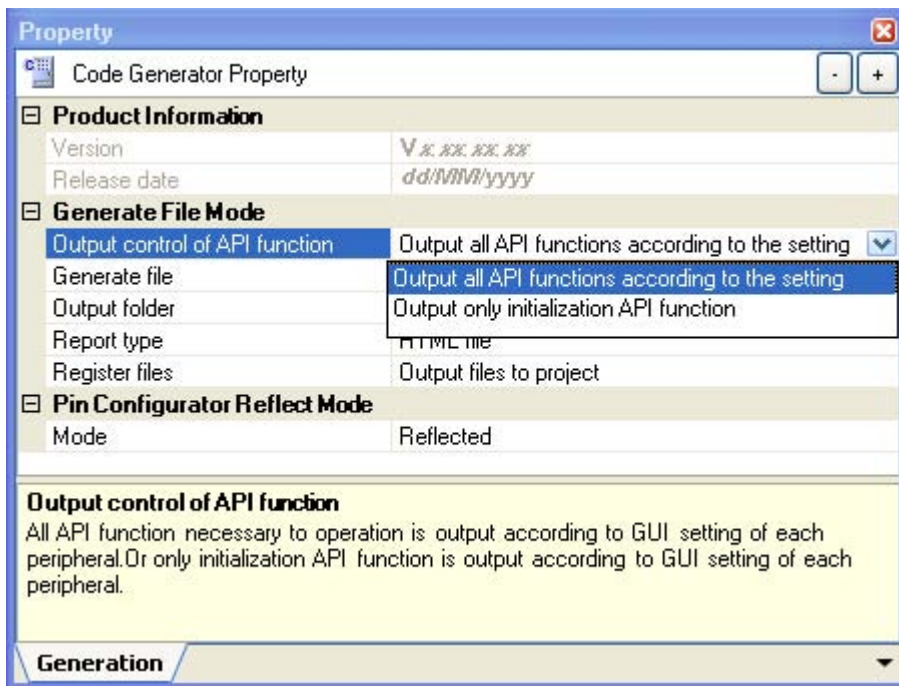
Figure 3-6. Configure Whether to Register



3.5.1 Setting that determines whether or not to generate source code

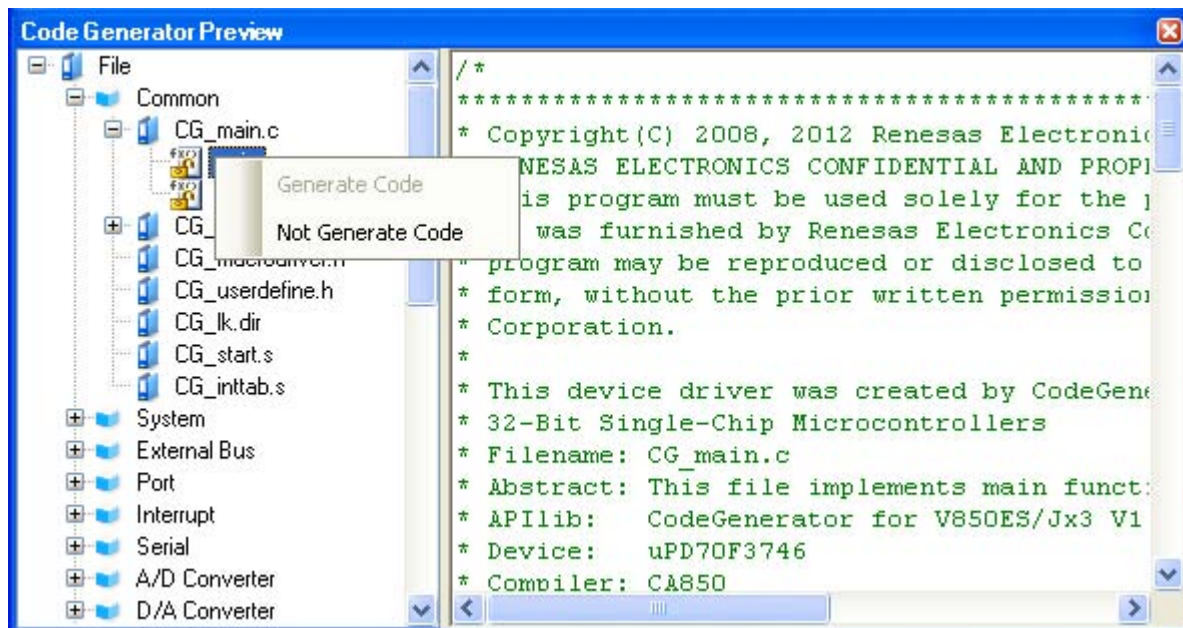
You can set the type of output API functions (all API functions or only initialization API functions) by selecting [Output all API function according to the setting/Output only initialization API function] from [Generation] tab >> [Output control of API function] in the Property panel.

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](#).

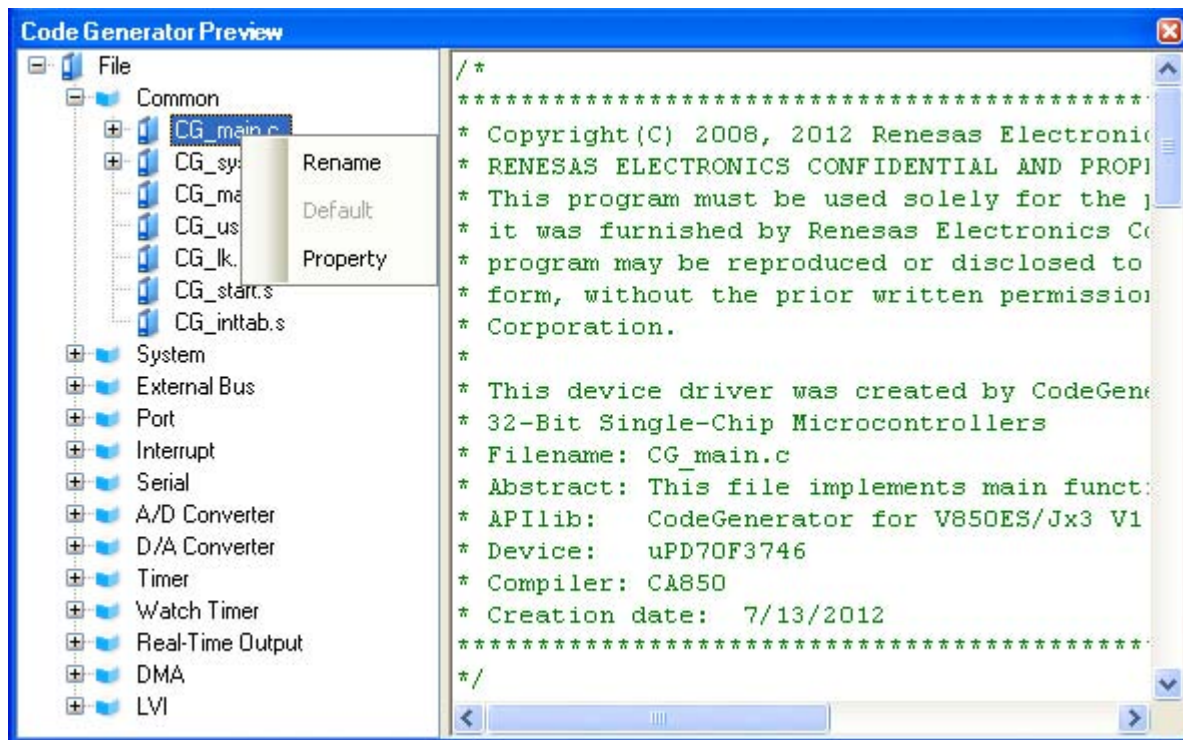
Table 3-4. Setting That Determines Whether or Not to Generate Source Code

| 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

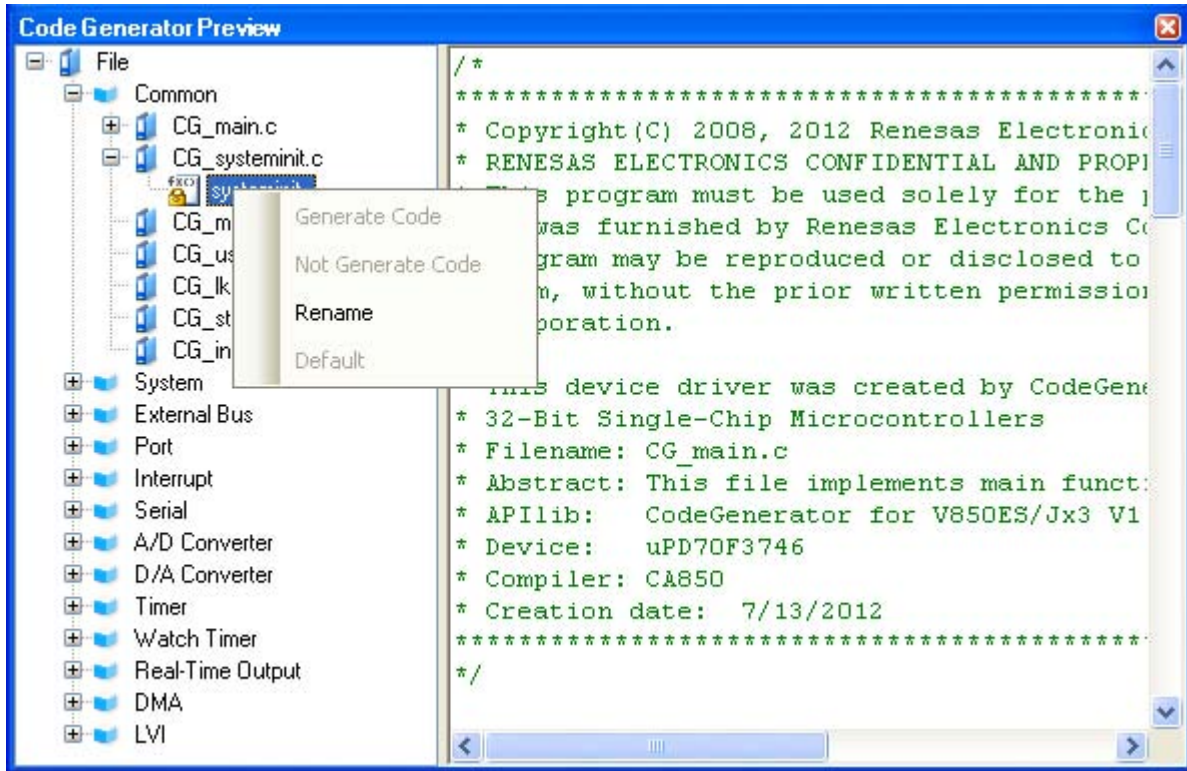


Remark To restore the default file name defined by the Code Generator, select [Default] from the context menu.

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

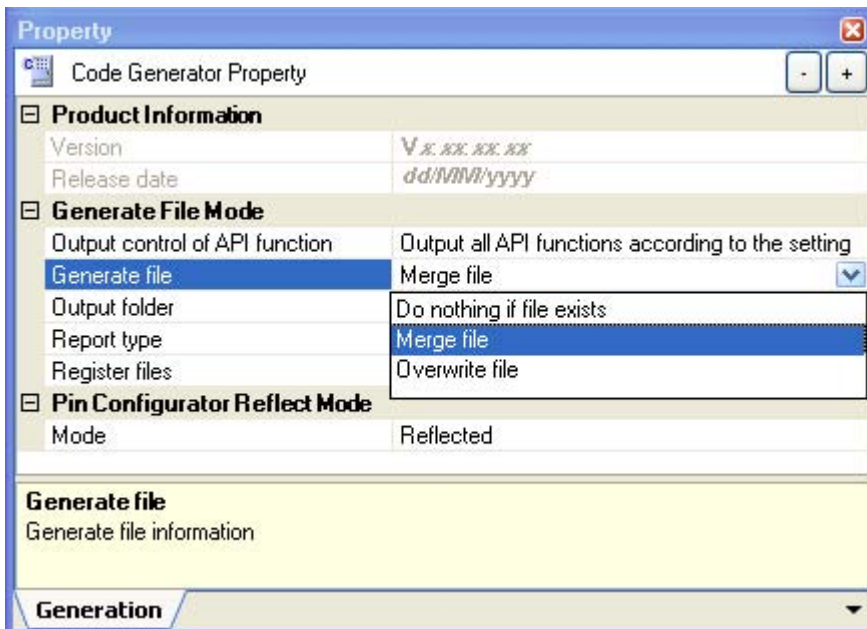


Remark To restore the default name of the API function defined by the Code Generator, select [Default] from the context menu.

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 the 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 the Code Generator |
| function | A file that contains the information regarding the source code |

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

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

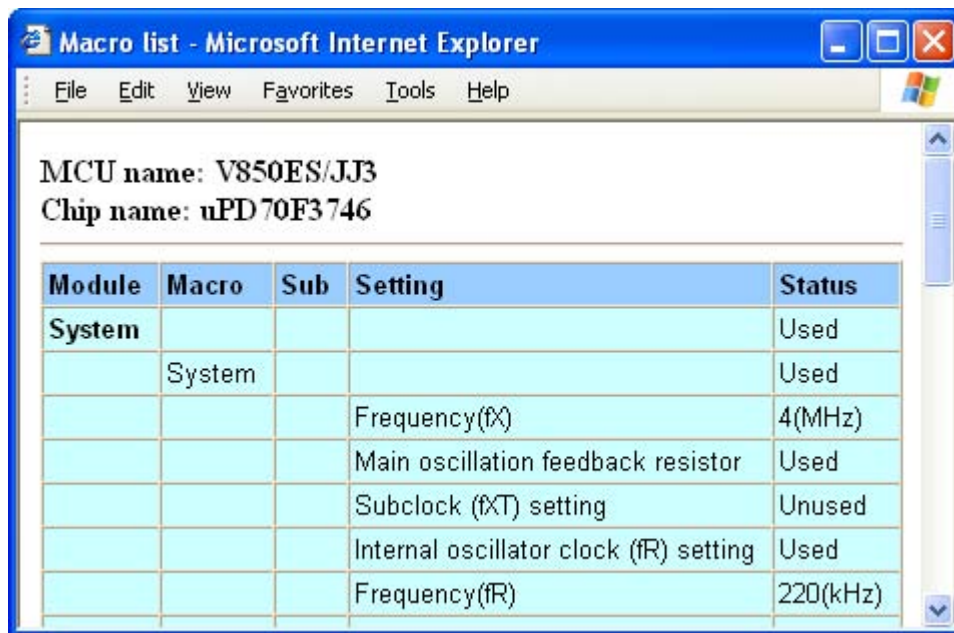


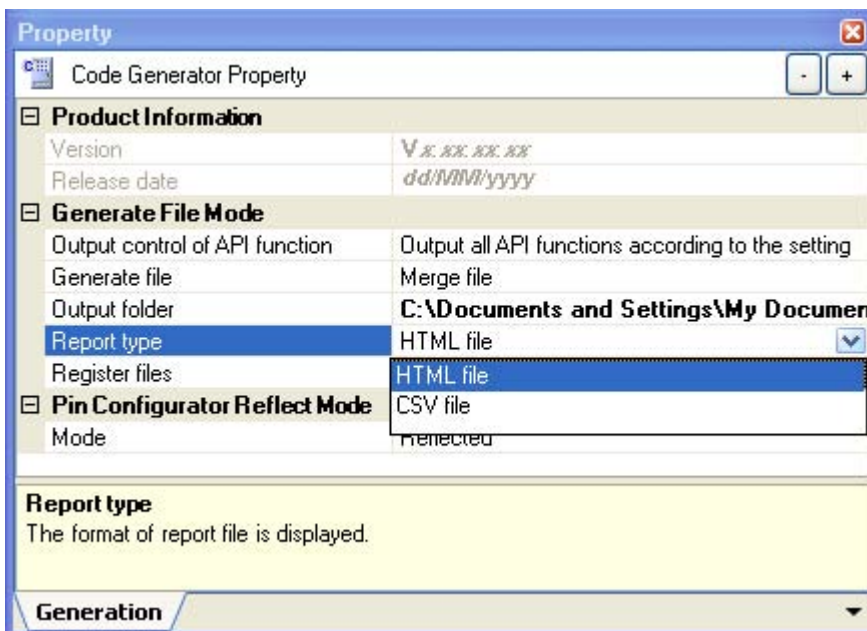
Figure 3-14. Output Example of Report File "function"

| 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 |
| | CG_macrodriver.h | | | CG_macrodriver.h | Used |
| | CG_userdefine.h | | | CG_userdefine.h | Used |
| | CG_lk.dir | | | CG_lk.dir | Used |
| | CG_start.s | | | CG_start.s | Used |
| | CG_inttab.s | | | CG_inttab.s | 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



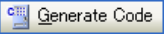
APPENDIX A WINDOW REFERENCE

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.

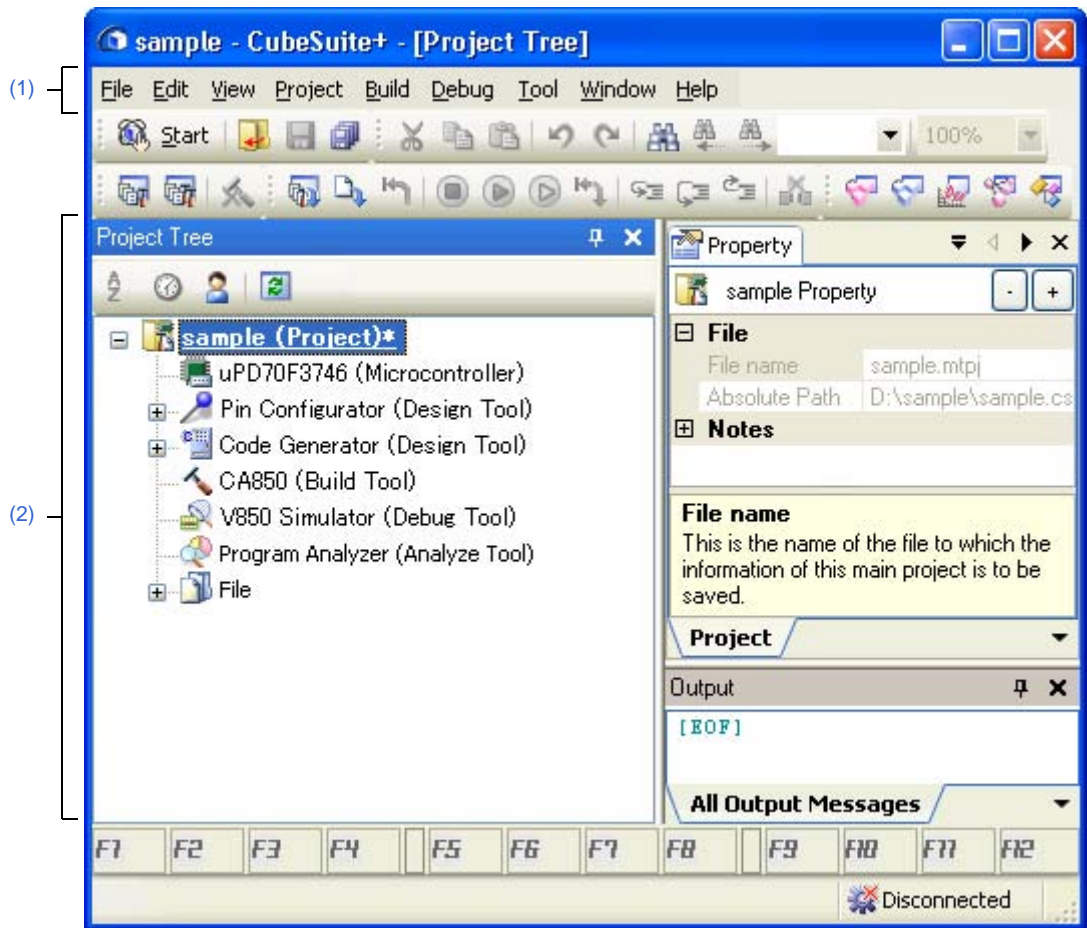
Table A-1. Window/Panel/Dialog Box List

| 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 | <p>Device Pin List panel-dedicated item</p> <p>Saves a report file (a file containing information configured using the Pin Configurator: device pin list) overwriting the existing file.</p> |
| Save Pin List As... | <p>Device Pin List panel-dedicated item</p> <p>Opens the Save As dialog box for naming and saving a report file (a file containing information configured using the Pin Configurator: device pin list).</p> |
| Save Top View | <p>Device Top View panel-dedicated item</p> <p>Saves a report file (a file containing information configured using the Pin Configurator: device top view) overwriting the existing file.</p> |
| Save Top View As... | <p>Device Top View panel-dedicated item</p> <p>Opens the Save As dialog box for naming and saving a report file (a file containing information configured using the Pin Configurator: device top view).</p> |
| Save Code Generator Report | <p>Code Generator panel/Code Generator Preview panel-dedicated item</p> <p>Outputs report files (a file containing information configured using the Code Generator and a file containing information regarding the source code).</p> <ul style="list-style-type: none"> - 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- <i>Tab Name</i> | <p>Output panel-dedicated item</p> <p>Saves the message corresponding to the specified tab overwriting the existing file.</p> |
| Save Output- <i>Tab Name</i> As... | <p>Output panel-dedicated item</p> <p>Opens the Save As dialog box for naming and saving the message corresponding to the specified tab.</p> |

(b) [Edit] menu

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

| | |
|------------|--|
| 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.

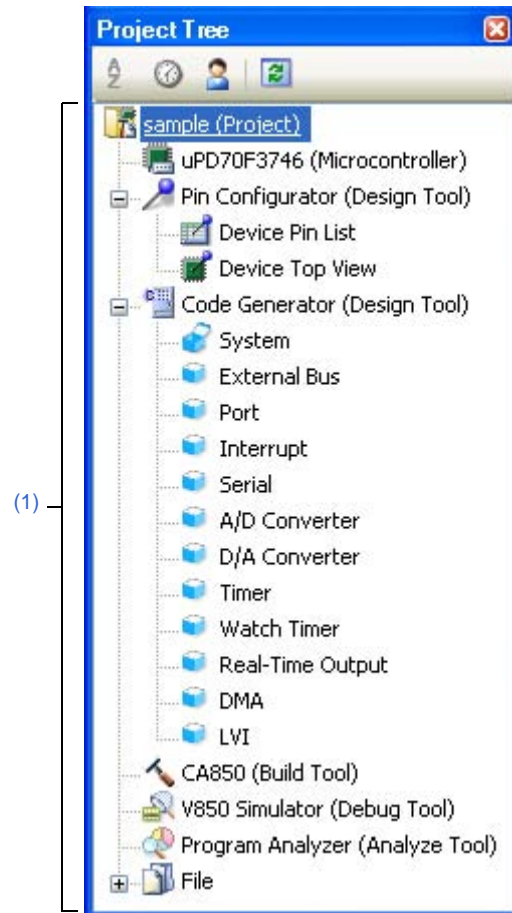
See the following sections for details on this area.

- [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 function button is not displayed.

| | |
|------------------|---|
| System | Opens the [System] for configuring the information necessary to control the functions of clock generator, on-chip debug function and functions of power-on-clear circuit provided by the microcontroller. |
| External Bus | Opens the [External Bus] for configuring the information necessary to control the functions of external bus interface (functions to connect an external bus to the area other than the built-in ROM, RAM or SFR) 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. |
| 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. |
| D/A Converter | Opens the [D/A Converter] for configuring the information necessary to control the function of D/A 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. |
| Watch Timer | Opens the [Watch Timer] for configuring the information necessary to control the functions of watch 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. |
| Real-Time Output | Opens the [Real-Time Output] for configuring the information necessary to control the real-time output functions provided by the microcontroller. |
| DMA | Opens the [DMA] for configuring the information necessary to control the functions of DMA (Direct Memory Access) 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 setting 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]

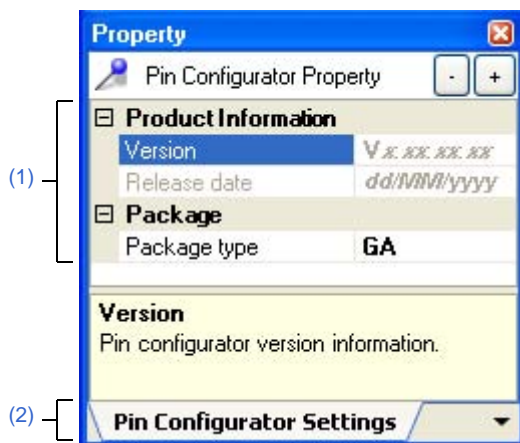
The following context menu items are displayed by right clicking the mouse.

| | |
|-----------------------|--|
| 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 (such as [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 (such as  ,  , 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 (such as [System], [Port], etc.)) selected in the [Project Tree panel](#), the peripheral function button (such as  ,  , 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. |

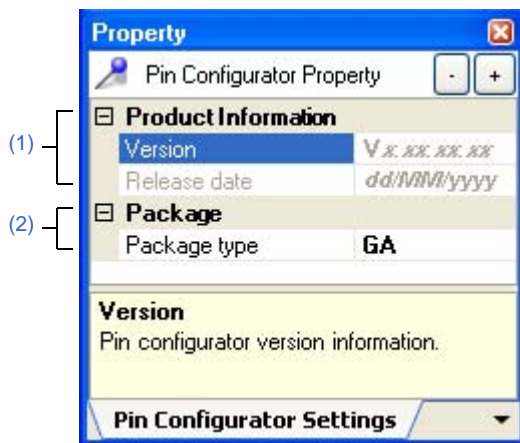
(2) While the item is not being edited

| | |
|-------------------------------|--|
| 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 the Pin Configurator.

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

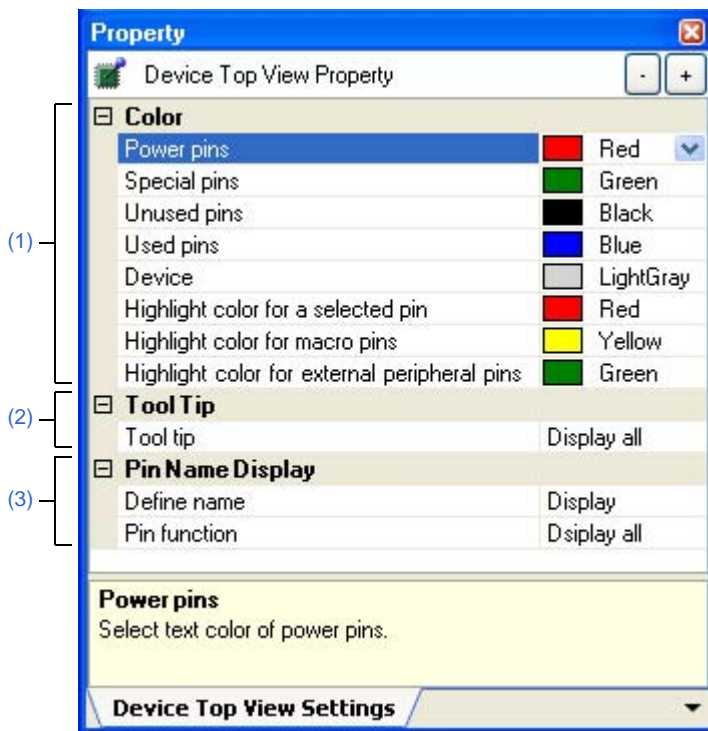
This area displays product information (Version and Release date) on Pin Configurator.

| | |
|--------------|---|
| 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

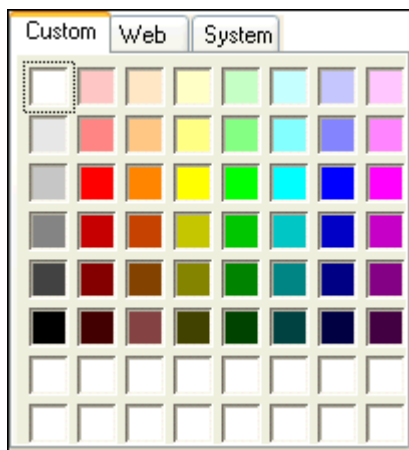
Select the display colors to differentiate the pin groups (Power pins, Special pins, etc.) in the device top view.

| | |
|--------------|--|
| 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. | |
| | 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" string 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

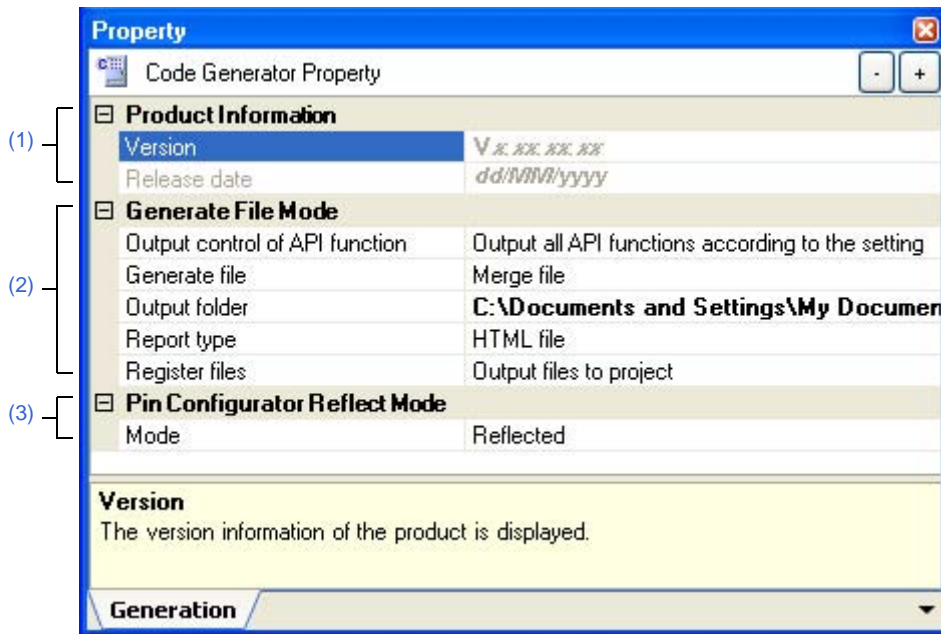
Select whether to display additional information about the pin in the device top view.

| | | |
|--------------|--|--|
| 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]

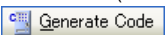
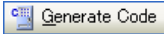

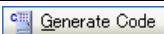
(1) [Product Information] category

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

| | |
|--------------|---|
| Version | Displays the version of the Code Generator (Code Library). |
| Release date | Displays the release date of the 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 the 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  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 select the operation mode applied when the  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 <code>/* Start user code ...</code> . Do not edit comment generated here <code>*/</code> and <code>/* End user code.</code> Do not edit comment generated here <code>*/</code> 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 select the destination folder for various files (source code and report files) which are output when the  button is pressed or when [File] menu >> [Save Code Generator Report] is selected. | |
| Report type | Views or select the format of the report files (a file containing information configured using the 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  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 the 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



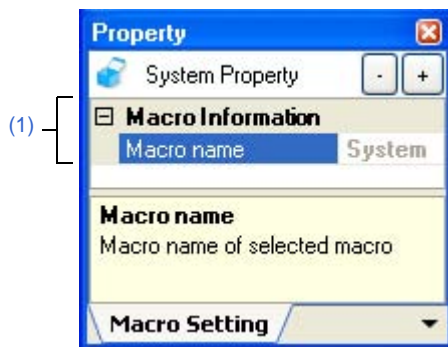
This tab allows you to view the information (Macro Information) on and change the setting for the peripheral function node (such as [System], [Port], etc.) selected in the [Project Tree panel](#), or the peripheral function button (such as , , etc.) pressed in the [Code Generator panel](#).

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 (such as [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 (such as , , etc.) in the [Code Generator panel](#) changes the content displayed accordingly.

[Description of each area]

(1) [Macro Information] category

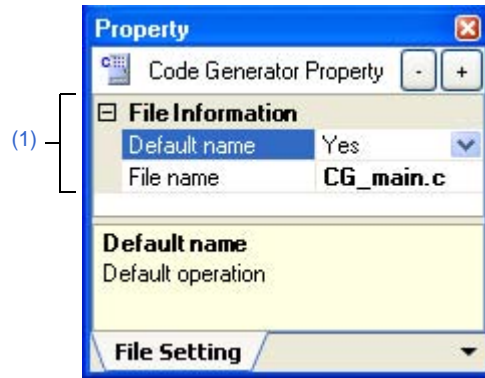
This area allows you to view the information (Macro name) on and change the setting for the peripheral function node (such as [System], [Port], etc.) selected in the [Project Tree panel](#), or the peripheral function button pressed in the [Code Generator panel](#).

| | |
|------------|--|
| 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](#).

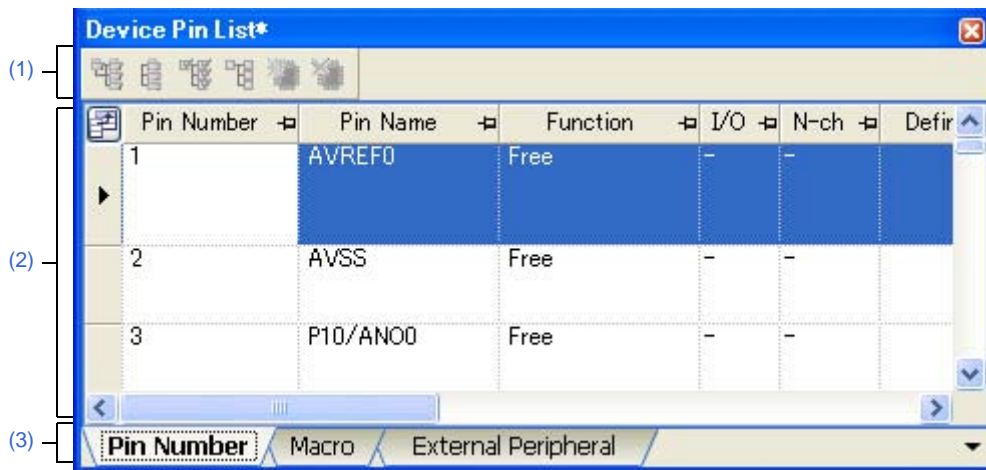
| | | |
|--------------|---|---|
| Default name | Views or select the setting that determines whether the name of the file selected in the Code Generator Preview panel is a default name or not. | |
| | Yes | The file name is a default name. Changing this area from "No" to "Yes" changes the name of the 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 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

Display 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 the 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 the Pin Configurator: device pin list). |

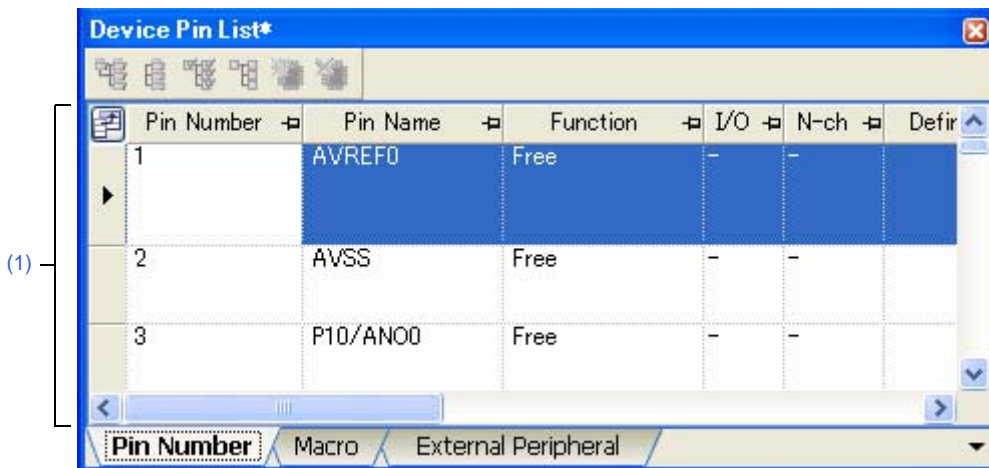
[[Help] menu (Device Pin List panel-dedicated items)]

| | |
|-------------------------------------|----------------------------------|
| 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

Display 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. |

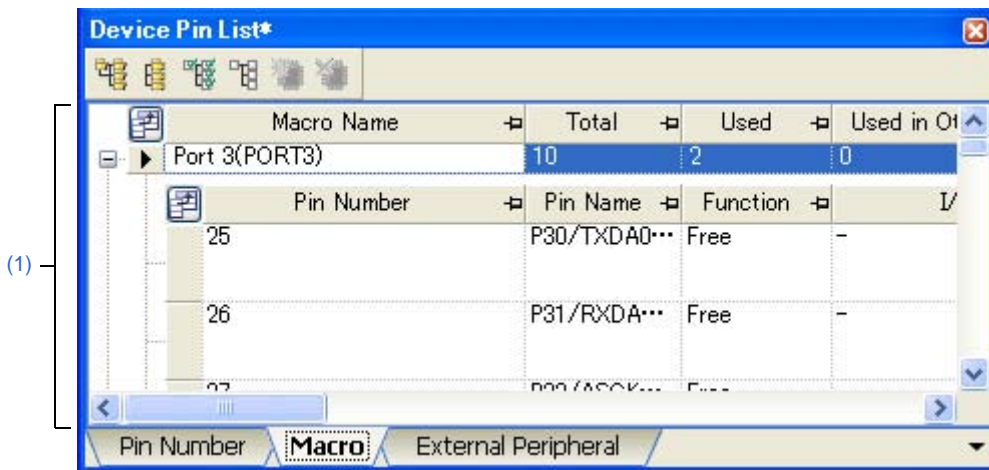
| Column Heading | Outline |
|---------------------------------|---|
| Define Name | This area allows you to assign a "user-defined pin name" to the pin. Within 256 characters can be entered in the [Define Name]. |
| Description | Displays the summary of function of the pin. |
| Recommend Connection 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

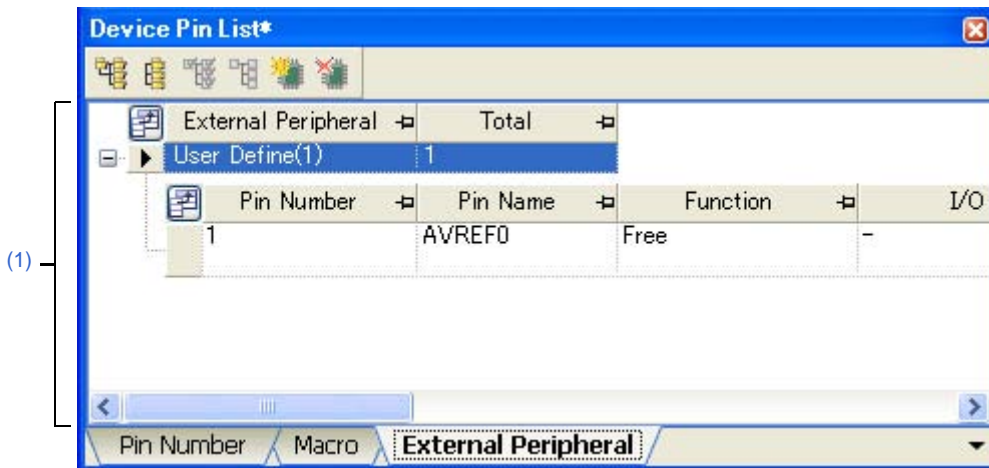
| Column Heading | Outline |
|---------------------------------|---|
| Pin Number | Displays the pin number of the pin. |
| Pin Name | Displays the pin name of the pin. |
| Function | This area allows you to select "which function to use" when the pin has more than one functions. |
| I/O | This area 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

Display 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. |

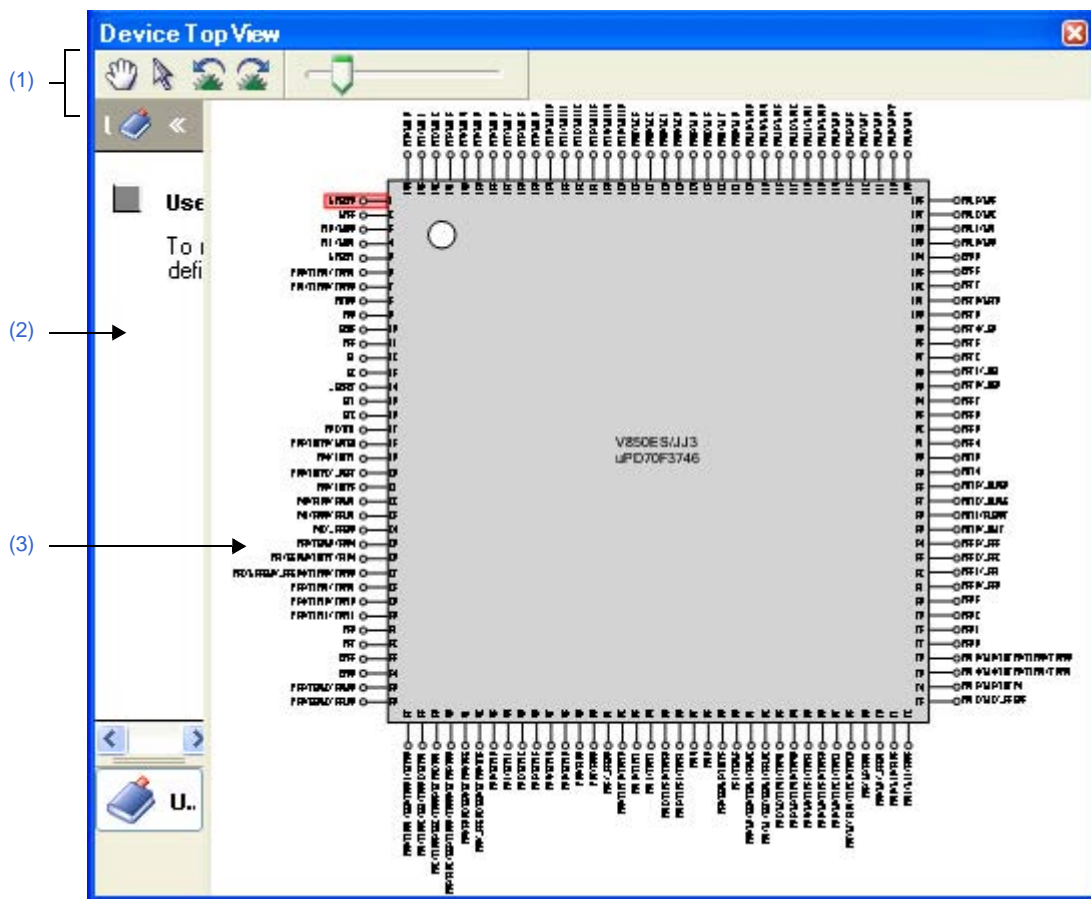
- Remarks 1.** You cannot add information in the "External Peripheral Name", "Connected Pins", "Pin Number", "Pin Name", "Description", "Recommend Connection for Unused" and "Attention" columns because they contain fixed information.
- If the "Free" in the "Function" column is changed to a specific pin 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](#).
 - To move columns (change the display order) in the device pin list, drag and drop the desired column to the desired location.
 - 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 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 the 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 the 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. |

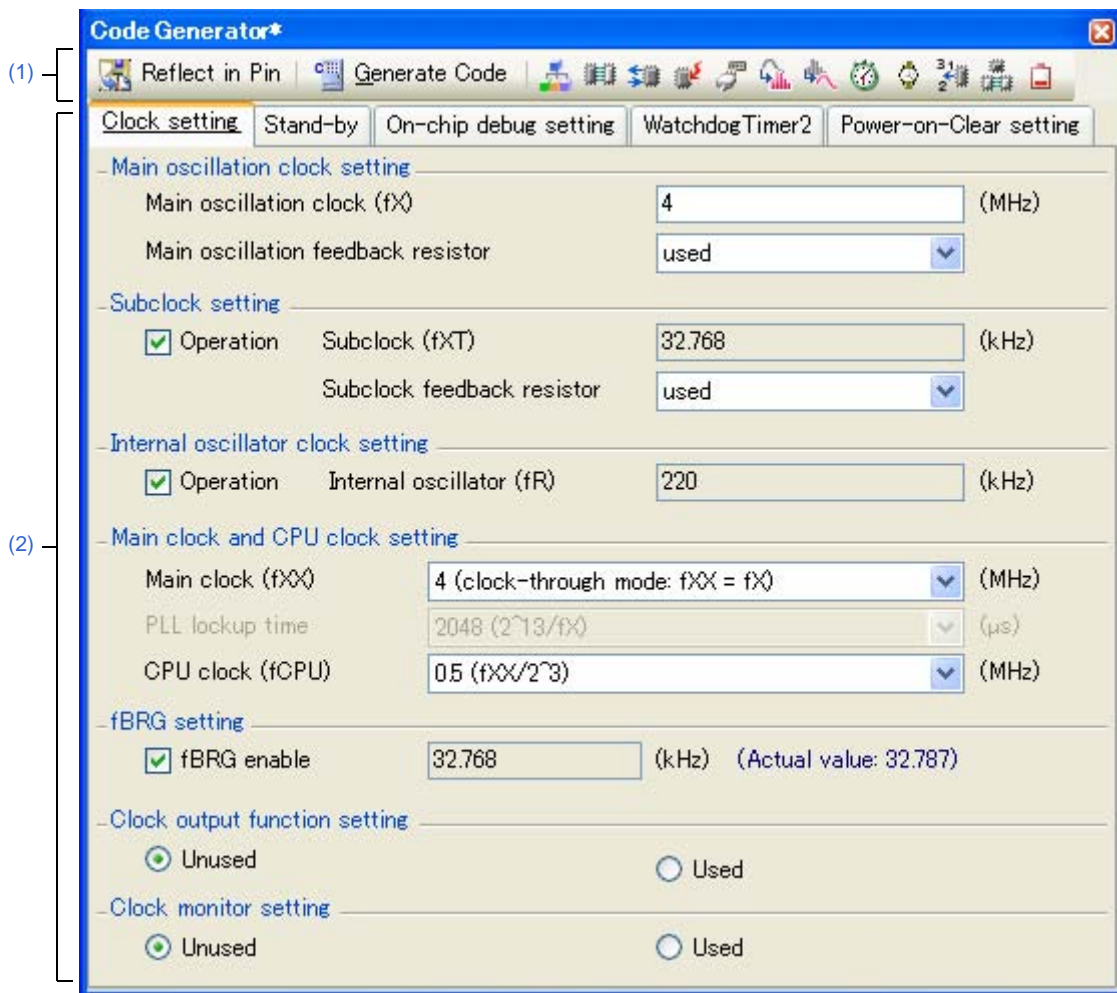
(2) When an external peripheral controller is right clicked

| | |
|----------------------------|---|
| 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 (such as  ,  , 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 function button is not displayed.

| | |
|--|--|
|  Combinate with PinPart | 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 the "Not reflected" is selected in the [PinPart Combination Mode] category of the [Generation] tab . |
|  Generate Code | Outputs the source code (device driver program) to the folder specified by selecting [Generation] tab >> [Output folder] in the Property panel . |
|  | Changes the view in the Information setting area to the "[System]" in order to configure the information required to control the clock generation function, standby function, and etc. provided by the microcontroller. |
|  | Changes the view in the Information setting area to the "[External Bus]" in order to configure the information required to control the external bus interface mode control function (function for connecting to external memory areas) 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 external interrupt functions and the key interrupt function provided by the microcontroller". |
|  | Changes the view in the Information setting area to the "[Serial]" in order to configure the information required to control the asynchronous serial interface A (UARTA), 3-wire variable-length serial I/O (CSIB), and etc. 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 "[D/A Converter]" for configuring the information necessary to control the function of D/A converter provided by the microcontroller". |
|  | Changes the view in the Information setting area to the "[Timer]" in order to configure the information required to control the 16-bit timer/event counter P (TMP), 16-bit timer/event counter Q (TMQ), and 16-bit interval timer M (TMM) functions provided by the microcontroller. |
|  | Changes the content displayed in the Information setting area to the "[Watch 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 "[Real-Time Output]" 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 "[DMA] for configuring the information necessary to control the functions of DMA (Direct Memory Access) controller provided by the microcontroller". |
|  | Changes the content displayed in the Information setting area to the "[LVD] 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 the 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

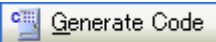
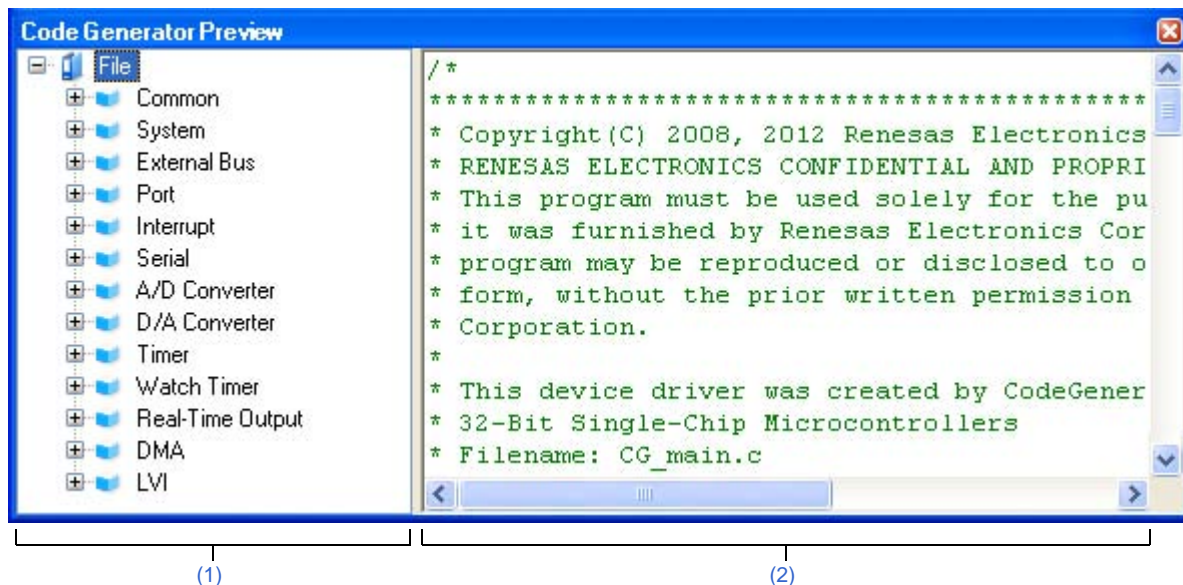
This panel allows you to confirm or configure on a per-API function basis the setting that determines whether or not the source code (device driver program) is generated when the  button is pressed in the [Code Generator panel](#). It also allows you to confirm the source code that reflects the information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

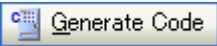
(2) Source code display area

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 the 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 | <p>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.</p> <p>Selecting this context menu item changes the icon of the currently selected API function from  to .</p> <p>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.</p> |
| Not generate code | <p>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.</p> <p>Selecting this context menu item changes the icon of the currently selected API function from  to .</p> |
| Rename | <p>Selecting this menu item changes the name portion of the currently selected file or API function into an edit box for editing the name.</p> <p>You can change the name of the file or API function by editing its name in the edit box.</p> |
| 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. |

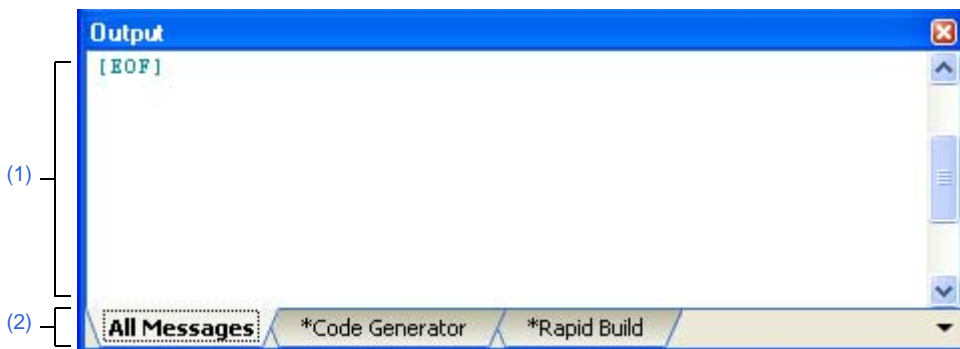
Output panel

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

The messages are classified by the message origination tool and displayed 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 | | Character color | Black | Displayed with information notices. |
| | | Background color | White | |
| Warning message | | Character color | Bule | Displayed with warnings about operations. |
| | | Background color | Standard color | |
| Error message | | 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- <i>Tab Name</i> | Saves the message corresponding to the specified tab overwriting the existing file. |
| Save Output- <i>Tab Name</i> 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]

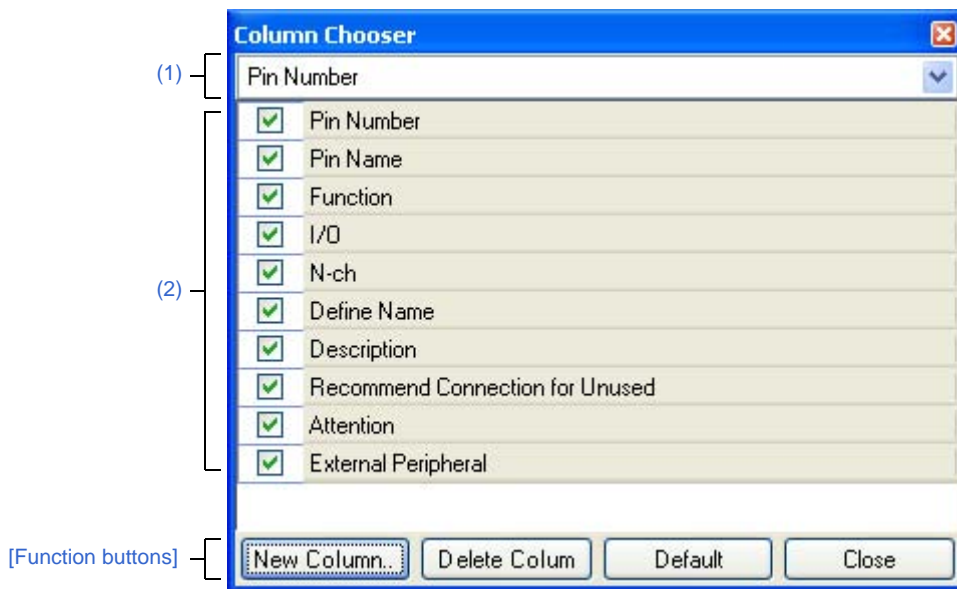
The following context menu items are displayed by right clicking the mouse.

| | |
|-----------------------|--|
| 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 | <p>Cancels the search currently being executed.</p> <p>This is invalid when a search is not being executed.</p> |
| Open Help for Message | <p>Displays help for the message on the current caret location.</p> <p>This only applies to warning messages and error messages.</p> |

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)

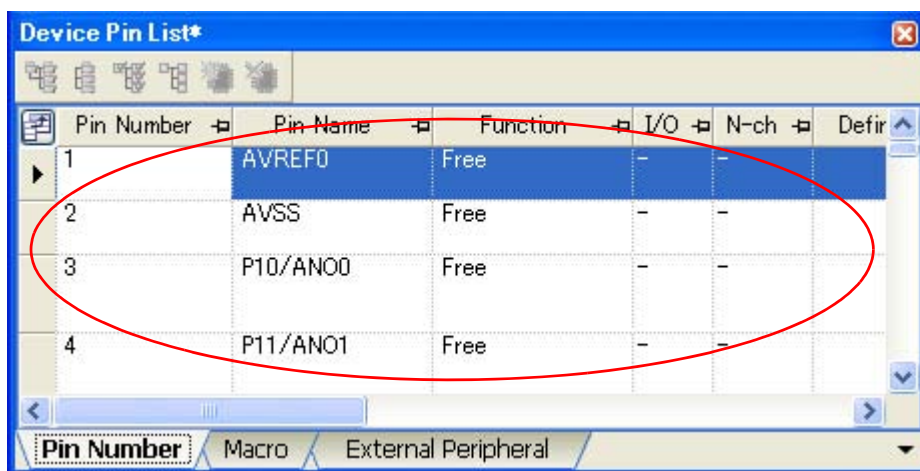


Figure A-21. Operational Object ([Macro] Tab: First Layer)

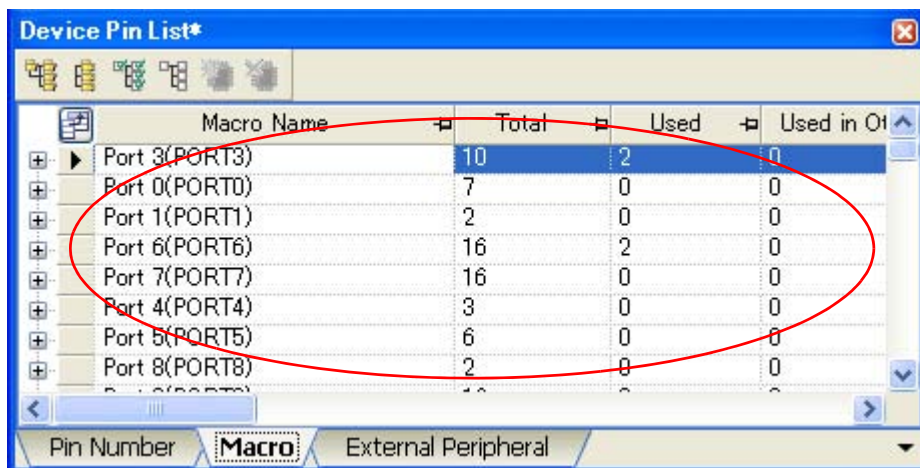


Figure A-22. Operational Object ([Macro] Tab: Second Layer)

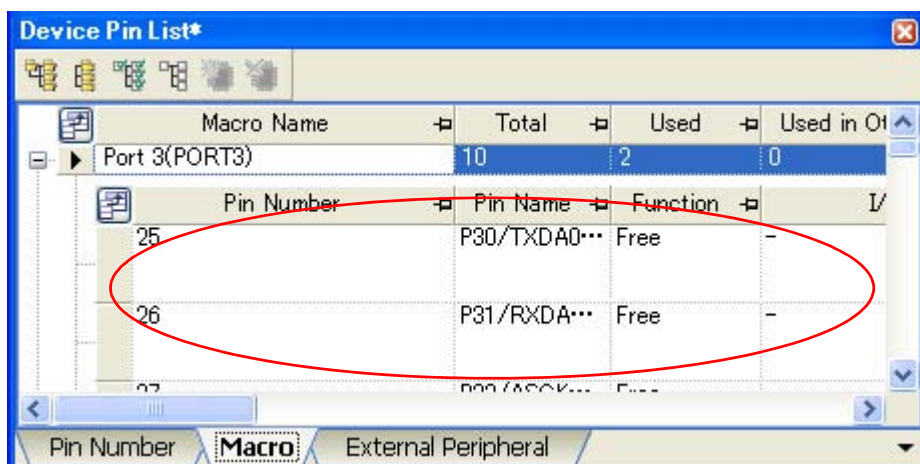


Figure A-23. Operational Object ([External Peripheral] Tab: First Layer)

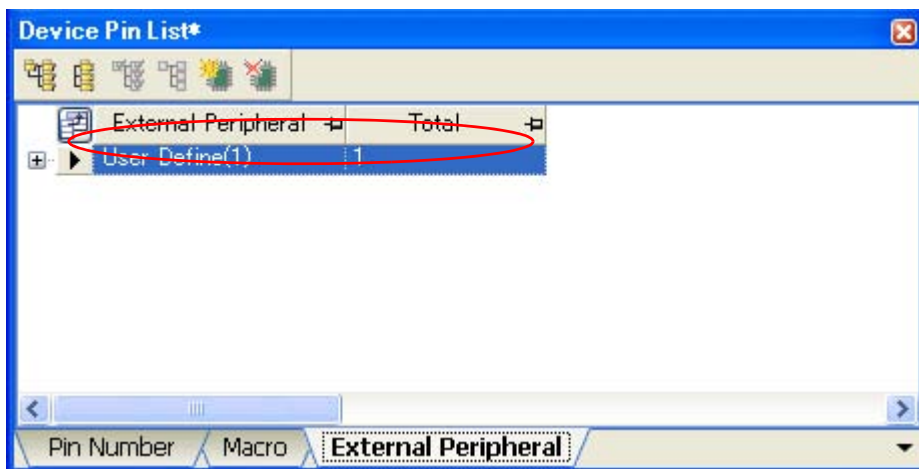
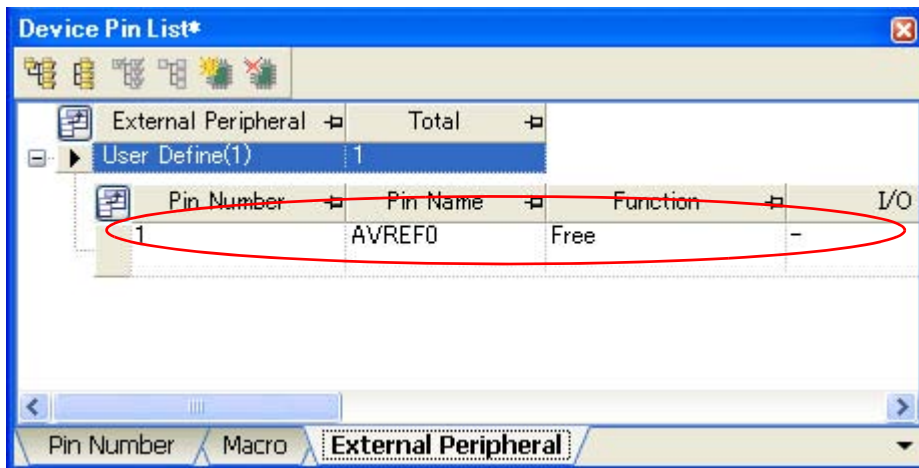


Figure A-24. Operational Object ([External Peripheral] Tab: Second Layer)



(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. |

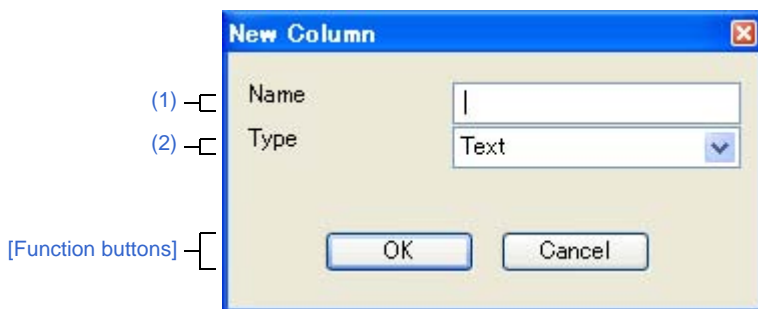
[Function buttons]

| 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. |
| Check 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.

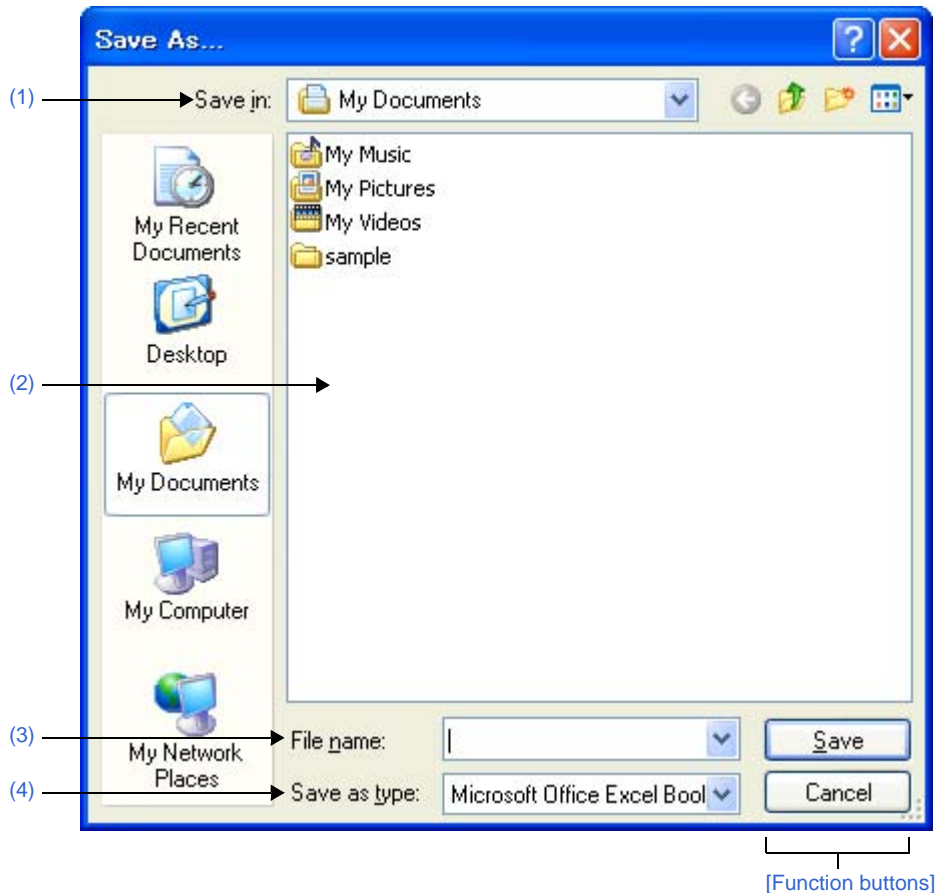
[Function buttons]

| 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) | PMG 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 | <i>PeripheralFunctionName.c</i> | Initial function, API function |
| | <i>PeripheralFunctionName_user.c</i> | Interrupt function (MD_INTxxx), callback function |
| | <i>PeripheralFunctionName.h</i> | 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.dir | Link directive |
| | CG_start.s | Startup routine |
| | CG_inttab.s | Interrupt vector table |

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_SelectPIIMode CG_SelectSSCGMode WDT2_Restart CRC_Start CRC_SetData CRC_GetResult |
| | CG_system_user.c | CLOCK_UserInit CG_ReadResetSource MD_INTWDT2 |
| | CG_system.h | - |
| External Bus | CG_bus.c | BUS_Init |
| | CG_bus_user.c | BUS_UserInit |

| Peripheral Function | Source File Name | Names of API Functions Included |
|---------------------|------------------|---|
| External Bus | CG_bus.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 INTPn_Enable KEY_Disable KEY_Enable |
| | CG_int_user.c | INTP_UserInit KEY_UserInit MD_INTNMI MD_INTPn MD_INTKR |
| | CG_int.h | - |
| Serial | CG_serial.c | UARTAn_Init UARTAn_Start UARTAn_Stop UARTAn_SendData UARTAn_ReceiveData UARTBn_Init UARTBn_Start UARTBn_Stop UARTBn_SendData UARTBn_ReceiveData UARTCn_Init UARTCn_Start UARTCn_Stop UARTCn_SendData UARTCn_ReceiveData CSIBn_Init CSIBn_Start CSIBn_Stop CSIBn_SendData CSIBn_ReceiveData CSIBn_SendReceiveData CSIEn_Init CSIEn_Start CSIEn_Stop CSIEn_SendData CSIEn_ReceiveData CSIEn_SendReceiveData |

| Peripheral Function | Source File Name | Names of API Functions Included |
|---------------------|------------------|--|
| Serial | CG_serial.c | CSIFn_Init CSIFn_Start CSIFn_Stop CSIFn_SendData CSIFn_ReceiveData IIC0n_Init IIC0n_Stop IIC0n_StopCondition IIC0n_MasterSendStart IIC0n_MasterReceiveStart IIC0n_SlaveSendStart IIC0n_SlaveReceiveStart |
| | CG_serial_user.c | UARTAn_UserInit UARTAn_SendEndCallback UARTAn_ReceiveEndCallback UARTAn_ErrorCallback UARTAn_SoftOverRunCallback UARTBn_UserInit UARTBn_SendEndCallback UARTBn_ReceiveEndCallback UARTBn_SingleErrorCallback UARTBn_FIFOErrorCallback UARTBn_TimeoutErrorCallback UARTBn_SoftOverRunCallback UARTCn_UserInit UARTCn_SendEndCallback UARTCn_ReceiveEndCallback UARTCn_ErrorCallback UARTCn_SoftOverRunCallback CSIBn_UserInit CSIBn_SendEndCallback CSIBn_ReceiveEndCallback CSIBn_ErrorCallback CSIEn_UserInit CSIEn_SendEndCallback CSIEn_ReceiveEndCallback CSIEn_ErrorCallback CSIFn_UserInit CSIFn_SendEndCallback CSIFn_ReceiveEndCallback CSIFn_ErrorCallback IIC0n_UserInit IIC0n_MasterSendEndCallback IIC0n_MasterReceiveEndCallback IIC0n_MasterErrorCallback IIC0n_SlaveSendEndCallback IIC0n_SlaveReceiveEndCallback IIC0n_SlaveErrorCallback |

| Peripheral Function | Source File Name | Names of API Functions Included |
|---------------------|------------------|---|
| Serial | CG_serial_user.c | IIC0n_GetStopConditionCallback MD_INTUAnT MD_INTUAnR MD_INTUBnTIT MD_INTUBnTIF MD_INTUBnTIR MD_INTUBnTIRE MD_INTUBnTITO MD_INTUCnT MD_INTUCnR MD_INTCBnT MD_INTCBnR MD_INTCEnT MD_INTCEnTIOF MD_INTCFnT MD_INTCFnR MD_INTIICn |
| | CG_serial.h | - |
| A/D Converter | CG_ad.c | AD_Init AD_Start AD_Stop AD_SelectADChannel AD_SetPFTCondition AD_Read AD_ReadByte |
| | CG_ad_user.c | AD_UserInit MD_INTAD |
| | CG_ad.h | - |
| D/A Converter | CG_da.c | DAn_Init DAn_Start DAn_Stop DAn_SetValue |
| | CG_da_user.c | DAn_UserInit |
| | CG_da.h | - |
| Timer | CG_timer.c | TMPn_Init TMPn_Start TMPn_Stop TMPn_ChangeTimerCondition TMPn_GetPulseWidth TMPn_GetFreeRunningValue TMPn_ChangeDuty TMPn_SoftwareTriggerOn TMQ0_Init TMQ0_Start TMQ0_Stop |

| Peripheral Function | Source File Name | Names of API Functions Included |
|---------------------|------------------|---|
| Timer | CG_timer.c | TMQ0_ChangeTimerCondition TMQ0_GetPulseWidth TMQ0_GetFreeRunningValue TMQ0_ChangeDuty TMQ0_SoftwareTriggerOn TAA _n _Init TAA _n _Start TAA _n _Stop TAA _n _ChangeTimerCondition TAA _n _ControlOutputToggle TAA _n _GetPulseWidth TAA _n _GetFreeRunningValue TAA _n _ChangeDuty TAA _n _SoftwareTriggerOn TAB _n _Init TAB _n _Start TAB _n _Stop TAB _n _ChangeTimerCondition TAB _n _ControlOutputToggle TAB _n _GetPulseWidth TAB _n _GetFreeRunningValue TAB _n _ChangeDuty TAB _n _SoftwareTriggerOn TMT0_Init TMT0_Start TMT0_Stop TMT0_ChangeTimerCondition TMT0_GetPulseWidth TMT0_GetFreeRunningValue TMT0_ChangeDuty TMT0_SoftwareTriggerOn TMT0_EnableHold TMT0_DisableHold TMT0_ChangeCountValue TMM _n _Init TMM _n _Start TMM _n _Stop TMM _n _ChangeTimerCondition |
| | CG_timer_user.c | TMP _n _UserInit TMQ0_UserInit TAA _n _UserInit TAB _n _UserInit TMT0_UserInit TMM _n _UserInit MD_INTTP _n OV MD_INTTP _n CC _m MD_INTTQ0OV MD_INTTQ0CC _m |

| Peripheral Function | Source File Name | Names of API Functions Included |
|---------------------|------------------|---|
| Timer | CG_timer_user.c | MD_INTTAA n OV MD_INTTAA n CC m MD_INTTTAB n OV MD_INTTAB n CC m MD_INTTT0EC MD_INTTT0OV MD_INTTT0CC m MD_INTTM n EQ0 |
| | CG_timer.h | - |
| Watch Timer | CG_wt.c | WT_Init WT_Start WT_Stop |
| | CG_wt_user.c | WT_UserInit MD_INTWT MD_INTWTI |
| | CG_wt.h | - |
| Real-time Clock | CG_rtc.c | RTC_Init 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_RC1CK1HZ_OutputEnable RTC_RC1CK1HZ_OutputDisable RTC_RC1CKO_OutputEnable RTC_RC1CKO_OutputDisable RTC_RC1CKDIV_OutputEnable RTC_RC1CKDIV_OutputDisable 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 |

| Peripheral Function | Source File Name | Names of API Functions Included |
|---------------------|------------------|---|
| Real-time Clock | CG_rtc_user.c | MD_INTRTCn |
| | CG_rtc.h | - |
| Real-Time Output | CG_rto.c | RTO _n _Init RTO _n _Enable RTO _n _Disable RTO _n _Set2BitData RTO _n _Set4BitData RTO _n _Set6BitData RTO _n _Set8BitData RTO _n _SetHigh2BitData RTO _n _SetLow2BitData RTO _n _SetHigh4BitData RTO _n _SetLow4BitData RTO _n _GetValue |
| | CG_rto_user.c | RTO _n _UserInit |
| | CG_rto.h | - |
| DMA | CG_dma.c | DMA _n _Init DMA _n _Enable DMA _n _Disable DMA _n _CheckStatus DMA _n _SetData DMA _n _SoftwareTriggerOn |
| | CG_dma_user.c | DMA _n _UserInit MD_INTDMA _n |
| | CG_dma.h | - |
| LVI | CG_lvi.c | LVI_Init LVI_InterruptModeStart LVI_ResetModeStart LVI_Start LVI_Stop |
| | 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 necessary to control clock functions. |
| | CLOCK_UserInit | Performs user-defined initialization relating to the clock. |
| | CG_ReadResetSource | Performs processing in response to a reset signal. |
| | CG_ChangeClockMode | Changes the CPU clock. |
| | CG_ChangeFrequency | Changes the CPU clock division ratio. |
| | CG_SelectPowerSaveMode | Configures the CPU's standby function. |
| | CG_SelectStabTime | Selects the oscillation stabilization time for the X1 oscillator. This will become necessary when STOP mode is released. |
| | CG_SelectPIIMode | Selects the operation mode of the PLL function. |
| | CG_SelectSSCGMode | Selects the operation mode of the SSCG (Spread Spectrum Clock Generator). |
| | WDT2_Restart | Clears the watchdog timer counter and resumes counting. |
| | CRC_Start | Begins detection of data-block errors. |
| | CRC_SetData | Sets data in the CRC input register (CRCIN). |
| | CRC_GetResult | Reads the results of the calculation stored in the CRC data register (CRCD). |
| External Bus | BUS_Init | Performs initialization necessary to control external bus interface functions (functions to connect an external bus to areas other than onboard ROM, ROM and RAM). |
| | BUS_UserInit | Performs user-defined initialization relating to the external bus interface. |
| 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. |

| Peripheral Function | API Function Name | Function |
|---------------------|-----------------------------|--|
| 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). |
| | 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 | UARTAn_Init | Performs initialization necessary to control the asynchronous serial interface A (UARTA) functions. |
| | UARTAn_UserInit | Performs user-defined initialization relating to the asynchronous serial interface A (UARTA). |
| | UARTAn_Start | Enables asynchronous serial interface A (UARTA). |
| | UARTAn_Stop | Disables asynchronous serial interface A (UARTA). |
| | UARTAn_SendData | Starts UARTAn data transmission. |
| | UARTAn_ReceiveData | Starts UARTAn data reception. |
| | UARTAn_SendEndCallback | Performs processing in response to the UARTAn consecutive transmission enable interrupt INTUAN T . |
| | UARTAn_ReceiveEndCallback | Performs processing in response to the UARTAn reception completion interrupt INTUAN R . |
| | UARTAn_ErrorCallback | Performs processing in response to the UARTAn reception error interrupt INTUAN R (overrun error, framing error, parity error). |
| | UARTAn_SoftOverRunCallback | Performs processing in response to detection of overrun error. |
| | UARTBn_Init | Performs initialization necessary to control the asynchronous serial interface B (UARTB) functions. |
| | UARTBn_UserInit | Performs user-defined initialization relating to the asynchronous serial interface B (UARTB). |
| | UARTBn_Start | Enables asynchronous serial interface B (UARTB). |
| | UARTBn_Stop | Disables asynchronous serial interface B (UARTB). |
| | UARTBn_SendData | Starts UARTBn data transmission. |
| | UARTBn_ReceiveData | Starts UARTBn data reception. |
| | UARTBn_SendEndCallback | Performs processing consequent to the transmission enable interrupt INTUBn TIT and the FIFO transmission completion interrupt INTUBn TIF . |

| Peripheral Function | API Function Name | Function |
|---------------------|-----------------------------|---|
| Serial | UARTBn_ReceiveEndCallback | Performs processing in response to the reception completion interrupt INTUBnTIR. |
| | UARTBn_SingleErrorCallback | Performs processing in response to the reception error interrupt INTUBnTIRE (overrun error, framing error, parity error). |
| | UARTBn_FIFOErrorCallback | Performs processing in response to the reception error interrupt INTUBnTIRE (overrun error, framing error, parity error). |
| | UARTBn_TimeoutErrorCallback | Performs processing in response to the reception timeout error interrupt INTUBnTITO. |
| | UARTBn_SoftOverRunCallback | Performs processing in response to detection of overrun error. |
| | UARTCn_Init | Performs initialization necessary to control the asynchronous serial interface A (UARTA) functions. |
| | UARTCn_UserInit | Performs user-defined initialization relating to the asynchronous serial interface A (UARTA). |
| | UARTCn_Start | Enables asynchronous serial interface A (UARTA). |
| | UARTCn_Stop | Disables asynchronous serial interface A (UARTA). |
| | UARTCn_SendData | Starts UARTAn data transmission. |
| | UARTCn_ReceiveData | Starts UARTAn data reception. |
| | UARTCn_SendEndCallback | Performs processing in response to the UARTAn consecutive transmission enable interrupt INTUANt. |
| | UARTCn_ReceiveEndCallback | Performs processing in response to the UARTAn reception completion interrupt INTUANr. |
| | UARTCn_ErrorCallback | Performs processing in response to the UARTAn reception error interrupt INTUANr (overrun error, framing error, parity error). |
| | UARTCn_SoftOverRunCallback | Performs processing in response to detection of overrun error. |
| | CSIBn_Init | Performs initialization necessary to control the 3-wire variable-length serial I/O B (CSIB) functions. |
| | CSIBn_UserInit | Performs user-defined initialization relating to the 3-wire variable-length serial I/O B (CSIB). |
| | CSIBn_Start | Enables 3-wire variable-length serial I/O B (CSIB). |
| | CSIBn_Stop | Disables 3-wire variable-length serial I/O B (CSIB). |
| | CSIBn_SendData | Starts CSIB data transmission. |
| | CSIBn_ReceiveData | Starts CSIB data reception. |
| | CSIBn_SendReceiveData | Starts CSIB data transmission/reception. |
| | CSIBn_SendEndCallback | Performs processing in response to the CSIBn reception completion interrupt INTCBnR or the CSIBn consecutive transmission write enable interrupt INTCBnT. |
| | CSIBn_ReceiveEndCallback | Performs processing in response to the CSIBn reception completion interrupt INTCBnR. |
| | CSIBn_ErrorCallback | Performs processing in response to the CSIBn reception error interrupt INTCBnR (overrun error). |
| | CSIEn_Init | Performs initialization necessary to control the 3-wire variable-length serial I/O E (CSIE) functions. |

| Peripheral Function | API Function Name | Function |
|---------------------|---|--|
| Serial | CSIE _n _UserInit | Performs user-defined initialization relating to the 3-wire variable-length serial I/O E (CSIE). |
| | CSIE _n _Start | Enables 3-wire variable-length serial I/O E (CSIE). |
| | CSIE _n _Stop | Disables 3-wire variable-length serial I/O E (CSIE). |
| | CSIE _n _SendData | Starts CSIE data transmission. |
| | CSIE _n _ReceiveData | Starts CSIE data reception. |
| | CSIE _n _SendReceiveData | Starts CSIE data transmission/reception. |
| | CSIE _n _SendEndCallback | Performs processing in response to the CSIE _n transmission/reception completion interrupt INTCE _n T. |
| | CSIE _n _ReceiveEndCallback | Performs processing in response to the CSIE _n transmission/reception completion interrupt INTCE _n T. |
| | CSIE _n _ErrorCallback | Performs processing in response to the CSIE _n BUF overflow interrupt INTCE _n TIOF. |
| | CSIF _n _Init | Performs initialization necessary to control the 3-wire variable-length serial I/O F (CSIF) functions. |
| | CSIF _n _UserInit | Performs user-defined initialization relating to the 3-wire variable-length serial I/O F (CSIF). |
| | CSIF _n _Start | Enables 3-wire variable-length serial I/O F (CSIF). |
| | CSIF _n _Stop | Disables 3-wire variable-length serial I/O F (CSIF). |
| | CSIF _n _SendData | Starts CSIF data transmission. |
| | CSIF _n _ReceiveData | Starts CSIF data reception. |
| | CSIF _n _SendReceiveData | Starts CSIF data transmission/reception. |
| | CSIF _n _SendEndCallback | Performs processing in response to the CSIF _n transmission/reception completion interrupt INTCF _n T. |
| | CSIF _n _ReceiveEndCallback | Performs processing in response to the CSIF _n transmission/reception completion interrupt INTCF _n T. |
| | CSIF _n _ErrorCallback | Performs processing in response to the CSIF _n reception error interrupt INTCF _n R (overrun error). |
| | IIC0 _n _Init | Performs initialization necessary to control the IIC bus functions. |
| | IIC0 _n _UserInit | Performs user-defined initialization relating to the IIC bus. |
| | IIC0 _n _Stop | Ends IIC0 _n communication. |
| | IIC0 _n _StopCondition | Generates a stop condition. |
| | IIC0 _n _MasterSendStart | Starts IIC0 _n master transmission. |
| | IIC0 _n _MasterReceiveStart | Starts IIC0 _n master reception. |
| | IIC0 _n _MasterSendEndCallback | Performs processing in response to the IIC _n master transfer completion interrupt INTIIC _n . |
| | IIC0 _n _MasterReceiveEndCallback | Performs processing in response to the IIC _n master transfer completion interrupt INTIIC _n . |
| | IIC0 _n _MasterErrorCallback | Performs processing in response to detection of error in IIC _n master communication. |
| | IIC0 _n _SlaveSendStart | Starts IIC0 _n slave transmission. |

| Peripheral Function | API Function Name | Function |
|---------------------|--------------------------------|--|
| Serial | IIC0n_SlaveReceiveStart | Starts IIC0n slave reception. |
| | IIC0n_SlaveSendEndCallback | Performs processing in response to the IICn slave transfer completion interrupt INTIICn. |
| | IIC0n_SlaveReceiveEndCallback | Performs processing in response to the IICn slave transfer completion interrupt INTIICn. |
| | IIC0n_SlaveErrorCallback | Performs processing in response to detection of error in IICn slave communication. |
| | IIC0n_GetStopConditionCallback | Performs processing in response to detection of stop condition. |
| 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_Start | Starts A/D conversion. |
| | AD_Stop | Ends A/D conversion. |
| | AD_SelectADChannel | Configures the analog voltage input pin for A/D conversion. |
| | AD_SetPFTCondition | Sets the information for operation in power-fail compare mode (comparison value and A/D conversion end interrupt INTAD trigger). |
| | 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). |
| D/A Converter | DAn_Init | Performs initialization necessary to control D/A converter functions. |
| | DAn_UserInit | Performs user-defined initialization relating to the D/A converter. |
| | DAn_Start | Starts D/A conversion. |
| | DAn_Stop | Ends D/A conversion. |
| | DAn_SetValue | Sets the analog voltage output to the ANOn pin. |
| Timer | TMPn_Init | Performs initialization necessary to control 16-bit timer/event counter P (TMP) functions. |
| | TMPn_UserInit | Performs user-defined initialization relating to the 16-bit timer/event counter P (TMP). |
| | TMPn_Start | Starts the count for 16-bit timer/event counter P (TMP). |
| | TMPn_Stop | Ends the count for 16-bit timer/event counter P (TMP). |
| | TMPn_ChangeTimerCondition | Changes the counter value for 16-bit timer/event counter P (TMP). |
| | TMPn_GetPulseWidth | Reads the pulse width of 16-bit timer/event counter P (TMP) (high/low level width). |
| | TMPn_GetFreeRunningValue | Reads the value captured by 16-bit timer/event counter P (TMP). |
| | TMPn_ChangeDuty | Changes the duty ratio of the PWM signal. |
| | TMPn_SoftwareTriggerOn | Generates the trigger (software trigger) for timer output. |

| Peripheral Function | API Function Name | Function |
|---------------------|--|---|
| Timer | TMQ0_Init | Performs initialization necessary to control 16-bit timer/event counter Q (TMQ) functions. |
| | TMQ0_UserInit | Performs user-defined initialization relating to the 16-bit timer/event counter Q (TMQ). |
| | TMQ0_Start | Starts the count for 16-bit timer/event counter Q (TMQ). |
| | TMQ0_Stop | Ends the count for 16-bit timer/event counter Q (TMQ). |
| | TMQ0_ChangeTimerCondition | Changes the counter value for 16-bit timer/event counter Q (TMQ). |
| | TMQ0_GetPulseWidth | Reads the pulse width of 16-bit timer/event counter Q (TMQ) (high/low level width). |
| | TMQ0_GetFreeRunningValue | Reads the value captured by 16-bit timer/event counter Q (TMQ). |
| | TMQ0_ChangeDuty | Changes the duty ratio of the PWM signal. |
| | TMQ0_SoftwareTriggerOn | Generates the trigger (software trigger) for timer output. |
| | TAA _n _Init | Performs initialization necessary to control 16-bit timer/event counter AA (TAA) functions. |
| | TAA _n _UserInit | Performs user-defined initialization relating to the 16-bit timer/event counter AA (TAA). |
| | TAA _n _Start | Starts the count for 16-bit timer/event counter AA (TAA). |
| | TAA _n _Stop | Ends the count for 16-bit timer/event counter AA (TAA). |
| | TAA _n _ChangeTimerCondition | Changes the counter value for 16-bit timer/event counter AA (TAA). |
| | TAA _n _ControlOutputToggle | Changes the toggle control of 16-bit timer/event counter AA (TAA). |
| | TAA _n _GetPulseWidth | Reads the pulse width of 16-bit timer/event counter AA (TAA) (high/low level width). |
| | TAA _n _GetFreeRunningValue | Reads the value captured by 16-bit timer/event counter AA (TAA). |
| | TAA _n _ChangeDuty | Changes the duty ratio of the PWM signal. |
| | TAA _n _SoftwareTriggerOn | Generates the trigger (software trigger) for timer output. |
| | TAB _n _Init | Performs initialization necessary to control 16-bit timer/event counter AB (TAB) functions. |
| | TAB _n _UserInit | Performs user-defined initialization relating to the 16-bit timer/event counter AB (TAB). |
| | TAB _n _Start | Starts the count for 16-bit timer/event counter AB (TAB). |
| | TAB _n _Stop | Ends the count for 16-bit timer/event counter AB (TAB). |
| | TAB _n _ChangeTimerCondition | Changes the counter value for 16-bit timer/event counter AB (TAB). |
| | TAB _n _ControlOutputToggle | Changes the toggle control of 16-bit timer/event counter AB (TAB). |
| | TAB _n _GetPulseWidth | Reads the pulse width of 16-bit timer/event counter AB (TAB) (high/low level width). |

| Peripheral Function | API Function Name | Function |
|---------------------|---------------------------|---|
| Timer | TABn_GetFreeRunningValue | Reads the value captured by 16-bit timer/event counter AB (TAB). |
| | TABn_ChangeDuty | Changes the duty ratio of the PWM signal. |
| | TABn_SoftwareTriggerOn | Generates the trigger (software trigger) for timer output. |
| | TMT0_Init | Performs initialization necessary to control 16-bit timer/event counter T (TMT) functions. |
| | TMT0_UserInit | Performs user-defined initialization relating to the 16-bit timer/event counter T (TMT). |
| | TMT0_Start | Starts the count for 16-bit timer/event counter T (TMT). |
| | TMT0_Stop | Ends the count for 16-bit timer/event counter T (TMT). |
| | TMT0_ChangeTimerCondition | Changes the counter value for 16-bit timer/event counter T (TMT). |
| | TMT0_GetPulseWidth | Reads the pulse width of 16-bit timer/event counter T (TMT) (high/low level width). |
| | TMT0_GetFreeRunningValue | Reads the value captured by 16-bit timer/event counter T (TMT). |
| | TMT0_ChangeDuty | Changes the duty ratio of the PWM signal. |
| | TMT0_SoftwareTriggerOn | Generates the trigger (software trigger) for timer output. |
| | TMT0_EnableHold | Changes the encoder counter control of 16-bit timer/event counter T (TMT) to holding count value. |
| | TMT0_DisableHold | Changes the encoder counter control of 16-bit timer/event counter T (TMT) to normal operation. |
| | TMT0_ChangeCountValue | Changes the initial counter value for 16-bit timer/event counter T (TMT). |
| | TMMn_Init | Performs initialization necessary to control 16-bit interval timer M (TMM) functions. |
| | TMMn_UserInit | Performs user-defined initialization relating to the 16-bit interval timer M (TMM). |
| | TMMn_Start | Starts the count for 16-bit interval timer M (TMM). |
| | TMMn_Stop | Ends the count for 16-bit interval timer M (TMM). |
| | TMMn_ChangeTimerCondition | Changes the counter value for 16-bit interval timer M (TMM). |
| Watch Timer | WT_Init | Performs initialization necessary to control watch timer functions. |
| | WT_UserInit | Performs user-defined initialization relating to the watch timer. |
| | WT_Start | Clears the watch timer counter and resumes counting. |
| | WT_Stop | Ends the count for watch timer. |
| 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_CounterEnable | Starts the count of the real-time counter (year, month, week-day, day, hour, minute, second). |

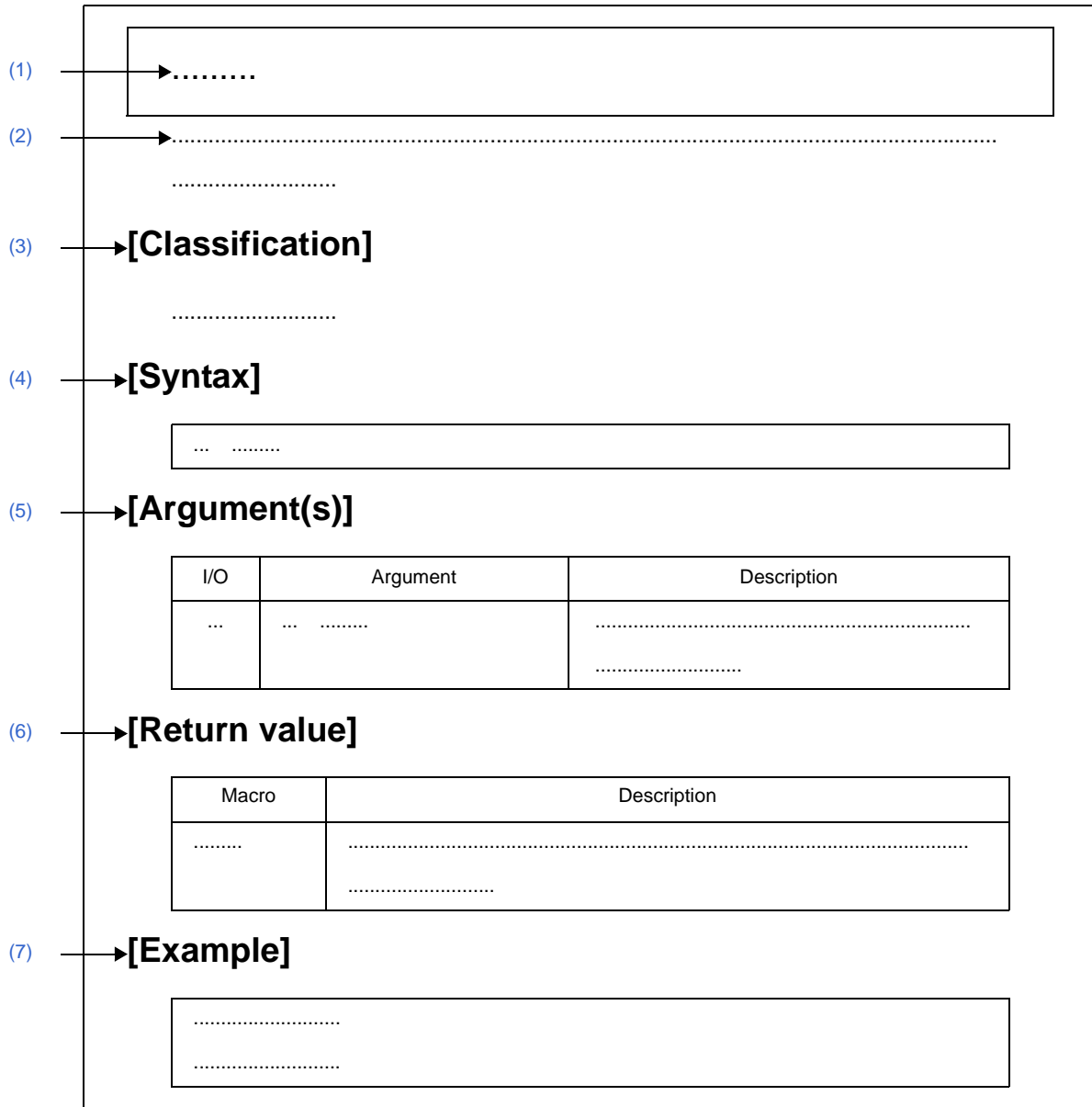
| Peripheral Function | API Function Name | Function |
|-------------------------|--|---|
| Real-time Clock | RTC_CounterDisable | Ends the count of the real-time counter (year, month, weekday, day, hour, minute, second). |
| | RTC_SetHourSystem | Sets the clock type (12-hour or 24-hour clock) of the 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 INTRTC0, then starts the cyclic interrupt function. |
| | RTC_ConstPeriodInterruptDisable | Ends the cyclic interrupt function. |
| | 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_IntervalStart | Starts the interval interrupt function. |
| | RTC_IntervalStop | Ends the interval interrupt function. |
| | RTC_IntervalInterruptEnable | Sets the cycle of the interrupts INTRTC2, then starts the interval interrupt function. |
| | RTC_IntervalInterruptDisable | Ends the interval interrupt function. |
| | RTC_RC1CK1HZ_OutputEnable | Enables output of the real-time counter correction clock (1 Hz) to the RC1CK1HZ pin. |
| | RTC_RC1CK1HZ_OutputDisable | Disables output of the real-time counter correction clock (1 Hz) to the RC1CK1HZ pin. |
| | RTC_RC1CKO_OutputEnable | Enables output of the real-time counter clock (32 kHz source) to the RC1CKO pin. |
| | RTC_RC1CKO_OutputDisable | Disables output of the real-time counter clock (32 kHz source) to the RC1CKO pin. |
| | RTC_RC1CKDIV_OutputEnable | Enables output of the real-time counter clock (32 kHz cycle) to the RC1CKDIV pin. |
| | RTC_RC1CKDIV_OutputDisable | Disables output of the real-time counter clock (32 kHz cycle) to the RC1CKDIV pin. |
| | 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. | |

| Peripheral Function | API Function Name | Function |
|---------------------|---------------------------|---|
| Real-time Clock | 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. |
| Real-Time Output | RTOn_Init | Performs initialization necessary to control real-time output functions. |
| | RTOn_UserInit | Performs user-defined initialization relating to the real-time output. |
| | RTOn_Enable | Enables (validates) real-time output. |
| | RTOn_Disable | Disables (invalidates) real-time output. |
| | RTOn_Set2BitData | Sets 2-bit data for real-time output. |
| | RTOn_Set4BitData | Sets 4-bit data for real-time output. |
| | RTOn_Set6BitData | Sets 6-bit data for real-time output. |
| | RTOn_Set8BitData | Sets 8-bit data for real-time output. |
| | RTOn_SetHigh2BitData | Sets higher 2-bit data for real-time output. |
| | RTOn_SetLow2BitData | Sets lower 2-bit data for real-time output. |
| | RTOn_SetHigh4BitData | Sets higher 4-bit data for real-time output. |
| | RTOn_SetLow4BitData | Sets lower 4-bit data for real-time output. |
| | RTOn_GetValue | Reads data from real-time output. |
| DMA | DMAn_Init | Performs initialization necessary to control DMA controller functions. |
| | DMAn_UserInit | Performs user-defined initialization relating to the DMA controller. |
| | DMAn_Enable | Enables operation of channel <i>n</i> . |
| | DMAn_Disable | Disables operation of channel <i>n</i> . |
| | DMAn_CheckStatus | Reads the transfer status (transfer complete/transfer ongoing). |
| | DMAn_SetData | Sets the RAM address of the transfer source/destination, and the number of times the data has been transferred. |
| | DMAn_SoftwareTriggerOn | Uses a software trigger as a DMA transfer start trigger. |
| 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_Start | Starts low-voltage detection. |
| | LVI_Stop | Stops low-voltage detection. |

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 necessary to control clock functions. |
| CLOCK_UserInit | Performs user-defined initialization relating to the clock. |
| CG_ReadResetSource | Performs processing in response to a reset signal. |
| CG_ChangeClockMode | Changes the CPU clock. |
| CG_ChangeFrequency | Changes the CPU clock division ratio. |
| CG_SelectPowerSaveMode | Configures the CPU's standby function. |
| CG_SelectStabTime | Selects the oscillation stabilization time for the X1 oscillator. This will become necessary when STOP mode is released. |
| CG_SelectPllMode | Selects the operation mode of the PLL function. |
| CG_SelectSSCGMode | Selects the operation mode of the SSCG (Spread Spectrum Clock Generator). |
| WDT2_Restart | Clears the watchdog timer counter and resumes counting. |
| CRC_Start | Begins detection of data-block errors. |
| CRC_SetData | Sets data in the CRC input register (CRCIN). |
| CRC_GetResult | Reads the results of the calculation stored in the CRC data register (CRCD). |

CLOCK_Init

Performs initialization necessary to control clock functions.

[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.

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 a 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 (); /* Perform process according to reset signal trigger */
    .....
}
```

[CG_system_user.c]

```
#include "CG_macrodriver.h"
void CG_ReadResetSource ( void ) {
    UCHAR resetflag = RESF; /* Reset control flag register: Obtain RESF contents */
    if ( resetflag & 0x1 ) { /* Trigger identification: Check LVIRF flag */
        ..... /* Process performed when low-voltage detector detects low voltage */
    } else if ( resetflag & 0x2 ) { /* Trigger identification: Check CLMRF flag */
        ..... /* Process performed when clock monitor oscillation stopped */
    } else if ( resetflag & 0x10 ) { /* Trigger identification: Check WDT2RF flag */
        ..... /* Process performed when watchdog timer 2 overflows */
    }
    .....
}
```

CG_ChangeClockMode

Changes the CPU 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 type MAINOSCCLK: Main clock oscillator (fXX) SUBCLK: Subclock oscillator (fXT) |

[Return value]

| Macro | Description |
|-------------|---|
| MD_OK | Normal completion |
| MD_ERROR1 | Exit with error (abend) - Cannot change from the subclock oscillator (fXT) to the main clock oscillator (fXX). |
| MD_ERROR2 | Exit with error (abend) - Cannot change from the main clock oscillator (fXX) to the subclock oscillator (fXT). |
| MD_ARGERROR | Invalid argument specification |

CG_ChangeFrequency

Changes the CPU clock division ratio.

[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 <i>clock</i> ; | Division ratio type SYSTEMCLOCK: fxx SYSONEHALF: fxx/2 SYSONEFOURTH: fxx/4 SYSONEEIGHTH: fxx/8 SYSONESIXTEENTH: fxx/16 SYSONETHIRTYSECOND: fxx/32 |

Remark "fxx" signifies the frequency of the main clock oscillator.

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ERROR | Exit with error (abend) |
| 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 [E/Sx3-H] [ES/Jx3-E] [ES/Jx3-H] PSSTOP: STOP mode PSHALT: HALT mode PSIDLE1: IDLE1 mode PSIDLE2: IDLE2 mode [ES/Jx3] [ES/Jx3-L] PSSTOP: STOP mode PSHALT: HALT mode |

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

CG_SelectStabTime

Selects the oscillation stabilization time for the X1 oscillator. This will become necessary when STOP mode is released.

[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 ¹⁰ /fx STLEVEL1: 2 ¹¹ /fx STLEVEL2: 2 ¹² /fx STLEVEL3: 2 ¹³ /fx STLEVEL4: 2 ¹⁴ /fx STLEVEL5: 2 ¹⁵ /fx STLEVEL6: 2 ¹⁶ /fx |

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

CG_SelectPllMode

Selects the operation mode of the PLL function.

[Classification]

CG_system.c

[Syntax]

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

[Argument(s)]

| I/O | Argument | Description |
|-----|-------------------------------|---|
| I | enum PllMode <i>pllmode</i> ; | Operation mode type [E/Sx3-H] [ES/Jx3] [ES/Jx3-L] SYSPLLOFF: Clock-through mode SYS4PLL: x4 (When PLL function is used) SYS8PLL: x8 (When PLL function is used) [ES/Jx3-E] [ES/Jx3-H] SYSPLLOFF: PLL stopped SYSPLLON: PLL operating |

[Return value]

| Macro | Description |
|-------------|--|
| MD_OK | Normal completion |
| MD_ERROR | Exit with error (abend) [ES/Jx3] [ES/Jx3-E] [ES/Jx3-H] [ES/Jx3-L] |
| MD_ERROR1 | Exit with error (abend) [E/Sx3-H] - Cannot change the operation mode. |
| MD_ERROR2 | Exit with error (abend) [E/Sx3-H] - Cannot change to the x4. |
| MD_ERROR3 | Exit with error (abend) [E/Sx3-H] - Cannot change to the x8. |
| MD_ARGERROR | Invalid argument specification |

CG_SelectSSCGMode

Selects the operation mode of the SSCG (Spread Spectrum Clock Generator).

[Classification]

CG_system.c

[Syntax]

```
#include "CG_macrodriver.h"
#include "CG_system.h"
MD_STATUS CG_SelectSSCGMode ( enum SSCGMode sscgmode );
```

[Argument(s)]

| I/O | Argument | Description |
|-----|---------------------------------|--|
| I | enum SSCGMode <i>sscgmode</i> ; | Operation mode type SYSSSCGON: SSCG operation enabled SYSSSCGOFF: SSCG operation stopped |

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ERROR | Exit with error (abend) |
| MD_ARGERROR | Invalid argument specification |

WDT2_Restart

Clears the watchdog timer counter and resumes counting.

[Classification]

CG_system.c

[Syntax]

```
void WDT2_Restart ( void );
```

[Argument(s)]

None.

[Return value]

None.

CRC_Start

Begins detection of data-block errors.

[Classification]

CG_system.c

[Syntax]

```
void CRC_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

CRC_SetData

Sets data in the CRC input register (CRCIN).

[Classification]

CG_system.c

[Syntax]

```
#include "CG_macrodriver.h"
void CRC_SetData ( UCHAR data );
```

[Argument(s)]

| I/O | Argument | Description |
|-----|---------------------|-------------|
| I | UCHAR <i>data</i> ; | Data to set |

[Return value]

None.

CRC_GetResult

Reads the results of the calculation stored in the CRC data register (CRCD).

[Classification]

CG_system.c

[Syntax]

```
#include "CG_macrodriver.h"
void CRC_GetResult ( USHORT *result );
```

[Argument(s)]

| I/O | Argument | Description |
|-----|-----------------|--|
| O | USHORT *result; | Pointer to area in which to store read calculation results |

[Return value]

None.

C.3.2 External Bus

Below is a list of API functions output by Code Generator for external bus interface use.

Table C-3. API Functions: [External Bus]

| API Function Name | Function |
|------------------------------|--|
| BUS_Init | Performs initialization necessary to control external bus interface functions (functions to connect an external bus to areas other than onboard ROM, ROM and RAM). |
| BUS_UserInit | Performs user-defined initialization relating to the external bus interface. |

BUS_Init

Performs initialization necessary to control external bus interface functions (functions to connect an external bus to areas other than onboard ROM, ROM and RAM).

[Classification]

CG_bus.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

BUS_UserInit

Performs user-defined initialization relating to the external bus interface.

Remark This API function is called as the [BUS_Init](#) callback routine.

[Classification]

CG_bus_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

C.3.3 Port

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

Table C-4. 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_ChangePmnInput

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

[Classification]

CG_port.c

[Syntax]

```
void PORT_ChangePmnInput ( void );
```

Remark *mn* is the port number.

[Argument(s)]

None.

[Return value]

None.

[Example]

Below is an example of switching the P00 pin's I/O mode from output mode to input mode.

[CG_main.c]

```
void main ( void ) {  
    .....  
    PORT_ChangeP00Input ( );          /* Switch I/O mode */  
    .....  
}
```

PORT_ChangePmnOutput

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_ChangePmnOutput ( BOOL initialvalue );
```

- [N-ch open drain output: yes]

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

Remark *nm* is the port number.

[Argument(s)]

| I/O | Argument | Description |
|-----|----------------------------|--|
| I | BOOL <i>enablench</i> ; | Output mode type MD_TRUE: N-ch open drain output (V _{DD} withstand voltage) mode MD_FALSE: Normal output mode |
| I | BOOL <i>initialvalue</i> ; | 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 (VDD 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.3.4 Interrupt

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

Table C-5. 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]

- [E/Sx3-H] [ES/Jx3-E] [ES/Jx3-H]

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

- [ES/Jx3] [ES/Jx3-L]

```
#include "CG_macrodriver.h"
#include "CG_int.h"
void 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]

- [E/Sx3-H] [ES/Jx3-E] [ES/Jx3-H]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

- [ES/Jx3] [ES/Jx3-L]

None.

[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    INTP $n$ _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    INTP $n$ _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.3.5 Serial

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

Table C-6. API Functions: [Serial]

| API Function Name | Function |
|-----------------------------|---|
| UARTAn_Init | Performs initialization necessary to control the asynchronous serial interface A (UARTA) functions. |
| UARTAn_UserInit | Performs user-defined initialization relating to the asynchronous serial interface A (UARTA). |
| UARTAn_Start | Enables asynchronous serial interface A (UARTA). |
| UARTAn_Stop | Disables asynchronous serial interface A (UARTA). |
| UARTAn_SendData | Starts UARTAn data transmission. |
| UARTAn_ReceiveData | Starts UARTAn data reception. |
| UARTAn_SendEndCallback | Performs processing in response to the UARTAn consecutive transmission enable interrupt INTUAnT. |
| UARTAn_ReceiveEndCallback | Performs processing in response to the UARTAn reception completion interrupt INTUAnR. |
| UARTAn_ErrorCallback | Performs processing in response to the UARTAn reception error interrupt INTUAnR (overrun error, framing error, parity error). |
| UARTAn_SoftOverRunCallback | Performs processing in response to detection of overrun error. |
| UARTBn_Init | Performs initialization necessary to control the asynchronous serial interface B (UARTB) functions. |
| UARTBn_UserInit | Performs user-defined initialization relating to the asynchronous serial interface B (UARTB). |
| UARTBn_Start | Enables asynchronous serial interface B (UARTB). |
| UARTBn_Stop | Disables asynchronous serial interface B (UARTB). |
| UARTBn_SendData | Starts UARTBn data transmission. |
| UARTBn_ReceiveData | Starts UARTBn data reception. |
| UARTBn_SendEndCallback | Performs processing consequent to the transmission enable interrupt INTUBnTIT and the FIFO transmission completion interrupt INTUBnTIF. |
| UARTBn_ReceiveEndCallback | Performs processing in response to the reception completion interrupt INTUBnTIR. |
| UARTBn_SingleErrorCallback | Performs processing in response to the reception error interrupt INTUBnTIRE (overrun error, framing error, parity error). |
| UARTBn_FIFOErrorCallback | Performs processing in response to the reception error interrupt INTUBnTIRE (overrun error, framing error, parity error). |
| UARTBn_TimeoutErrorCallback | Performs processing in response to the reception timeout error interrupt INTUBnTITO. |
| UARTBn_SoftOverRunCallback | Performs processing in response to detection of overrun error. |
| UARTCn_Init | Performs initialization necessary to control the asynchronous serial interface C (UARTC) functions. |
| UARTCn_UserInit | Performs user-defined initialization relating to the asynchronous serial interface C (UARTC). |
| UARTCn_Start | Enables asynchronous serial interface C (UARTC). |
| UARTCn_Stop | Disables asynchronous serial interface C (UARTC). |

| API Function Name | Function |
|----------------------------|---|
| UARTCn_SendData | Starts UARTCn data transmission. |
| UARTCn_ReceiveData | Starts UARTCn data reception. |
| UARTCn_SendEndCallback | Performs processing in response to the UARTCn consecutive transmission enable interrupt INTUCnT. |
| UARTCn_ReceiveEndCallback | Performs processing in response to the UARTCn reception completion interrupt INTUCnR. |
| UARTCn_ErrorCallback | Performs processing in response to the UARTCn reception error interrupt INTUCnR (overrun error, framing error, parity error). |
| UARTCn_SoftOverRunCallback | Performs processing in response to detection of overrun error. |
| CSIBn_Init | Performs initialization necessary to control the 3-wire variable-length serial I/O B (CSIB) functions. |
| CSIBn_UserInit | Performs user-defined initialization relating to the 3-wire variable-length serial I/O B (CSIB). |
| CSIBn_Start | Enables 3-wire variable-length serial I/O B (CSIB). |
| CSIBn_Stop | Disables 3-wire variable-length serial I/O B (CSIB). |
| CSIBn_SendData | Starts CSIB data transmission. |
| CSIBn_ReceiveData | Starts CSIB data reception. |
| CSIBn_SendReceiveData | Starts CSIB data transmission/reception. |
| CSIBn_SendEndCallback | Performs processing in response to the CSIBn reception completion interrupt INTCBnR or the CSIBn consecutive transmission write enable interrupt INTCBnT. |
| CSIBn_ReceiveEndCallback | Performs processing in response to the CSIBn reception completion interrupt INTCBnR. |
| CSIBn_ErrorCallback | Performs processing in response to the CSIBn reception error interrupt INTCBnR (overrun error). |
| CSIEn_Init | Performs initialization necessary to control the 3-wire variable-length serial I/O E (CSIE) functions. |
| CSIEn_UserInit | Performs user-defined initialization relating to the 3-wire variable-length serial I/O E (CSIE). |
| CSIEn_Start | Enables 3-wire variable-length serial I/O E (CSIE). |
| CSIEn_Stop | Disables 3-wire variable-length serial I/O E (CSIE). |
| CSIEn_SendData | Starts CSIE data transmission. |
| CSIEn_ReceiveData | Starts CSIE data reception. |
| CSIEn_SendReceiveData | Starts CSIE data transmission/reception. |
| CSIEn_SendEndCallback | Performs processing in response to the CSIEn transmission/reception completion interrupt INTCEnT. |
| CSIEn_ReceiveEndCallback | Performs processing in response to the CSIEn transmission/reception completion interrupt INTCEnT. |
| CSIEn_ErrorCallback | Performs processing in response to the CSIEnBUF overflow interrupt INTCEnTIOF. |
| CSIFn_Init | Performs initialization necessary to control the 3-wire variable-length serial I/O F (CSIF) functions. |
| CSIFn_UserInit | Performs user-defined initialization relating to the 3-wire variable-length serial I/O F (CSIF). |

| API Function Name | Function |
|--------------------------------|---|
| CSIFn_Start | Enables 3-wire variable-length serial I/O F (CSIF). |
| CSIFn_Stop | Disables 3-wire variable-length serial I/O F (CSIF). |
| CSIFn_SendData | Starts CSIF data transmission. |
| CSIFn_ReceiveData | Starts CSIF data reception. |
| CSIFn_SendReceiveData | Starts CSIF data transmission/reception. |
| CSIFn_SendEndCallback | Performs processing in response to the CSIFn transmission/reception completion interrupt INTCFnT. |
| CSIFn_ReceiveEndCallback | Performs processing in response to the CSIFn transmission/reception completion interrupt INTCFnT. |
| CSIFn_ErrorCallback | Performs processing in response to the CSIFn reception error interrupt INTCFnR (overrun error). |
| IIC0n_Init | Performs initialization necessary to control the IIC bus functions. |
| IIC0n_UserInit | Performs user-defined initialization relating to the IIC bus. |
| IIC0n_Stop | Ends IIC0n communication. |
| IIC0n_StopCondition | Generates a stop condition. |
| IIC0n_MasterSendStart | Starts IIC0n master transmission. |
| IIC0n_MasterReceiveStart | Starts IIC0n master reception. |
| IIC0n_MasterSendEndCallback | Performs processing in response to the IIC0n master transfer completion interrupt INTIICn. |
| IIC0n_MasterReceiveEndCallback | Performs processing in response to the IIC0n master transfer completion interrupt INTIICn. |
| IIC0n_MasterErrorCallback | Performs processing in response to detection of error in IIC0n master communication. |
| IIC0n_SlaveSendStart | Starts IIC0n slave transmission. |
| IIC0n_SlaveReceiveStart | Starts IIC0n slave reception. |
| IIC0n_SlaveSendEndCallback | Performs processing in response to the IIC0n slave transfer completion interrupt INTIICn. |
| IIC0n_SlaveReceiveEndCallback | Performs processing in response to the IIC0n slave transfer completion interrupt INTIICn. |
| IIC0n_SlaveErrorCallback | Performs processing in response to detection of error in IIC0n slave communication. |
| IIC0n_GetStopConditionCallback | Performs processing in response to detection of stop condition. |

UARTAn_Init

Performs initialization necessary to control the asynchronous serial interface A (UARTA) functions.

[Classification]

CG_serial.c

[Syntax]

```
void    UARTAn_Init ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTAn_UserInit

Performs user-defined initialization relating to the asynchronous serial interface A (UARTA).

Remark This API function is called as the [UARTAn_Init](#) callback routine.

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTAn_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTAn_Start

Enables asynchronous serial interface A (UARTA).

[Classification]

CG_serial.c

[Syntax]

```
void    UARTAn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTAn_Stop

Disables asynchronous serial interface A (UARTA).

[Classification]

CG_serial.c

[Syntax]

```
void    UARTAn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTAn_SendData

Starts UARTAn data transmission.

- Remarks 1.** This API function repeats the byte-level UARTAn transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.
- 2.** When performing a UARTAn transmission, [UARTAn_Start](#) must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|------------------------|---|
| I | UCHAR * <i>txbuf</i> ; | Pointer to a buffer storing the transmission data |
| I | USHORT <i>txnum</i> ; | Total amount of data to send |

[Return value]

| Macro | Description |
|---------------|--|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |
| MD_DATAEXISTS | Exit with error (abend) - Hold a next transmission data in the UA _n TX register. |

UARTAn_ReceiveData

Starts UARTAn data reception.

- Remarks 1.** This API function performs byte-level UARTAn reception the number of times specified by the parameter *rxnum*, and stores the data in the buffer specified by the parameter *rxbuf*.
- 2.** Actual UARTAn reception starts after this API function is called, and [UARTAn_Start](#) is then called.

[Classification]

CG_serial.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|------------------------|--|
| O | UCHAR * <i>rxbuf</i> ; | Pointer to a buffer to store the received data |
| I | USHORT <i>rxnum</i> ; | Total amount of data to receive |

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

UARTAn_SendEndCallback

Performs processing in response to the UARTAn consecutive transmission enable interrupt INTUA n T.

Remark This API function is called as the callback routine of interrupt process MD_INTUA n T corresponding to the UARTAn consecutive transmission enable interrupt INTUA n T (performed when total number of UARTAn transmissions specified by [UARTAn_SendData](#) parameter *txnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void UARTAn_SendEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTAn_ReceiveEndCallback

Performs processing in response to the UARTAn reception completion interrupt INTUANR.

Remark This API function is called as the callback routine of interrupt process MD_INTUANR corresponding to the UARTAn reception completion interrupt INTUANR (performed when total number of UARTAn receptions specified by [UARTAn_ReceiveData](#) parameter *rxnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void UARTAn_ReceiveEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTAn_ErrorCallback

Performs processing in response to the UARTAn reception error interrupt INTUAnR (overrun error, framing error, parity error).

Remark This API function is called as the callback routine of interrupt process MD_INTUAnR corresponding to the UARTAn reception error interrupt INTUAnR.

[Classification]

CG_serial_user.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|-------------------------|---|
| O | UCHAR <i>err_type</i> ; | Trigger for UARTAn reception error interrupt 00000xx1B: Overrun error 00000x1xB: Framing error 000001xxB: Parity error |

[Return value]

None.

UARTAn_SoftOverRunCallback

Performs processing in response to detection of overrun error.

Remark This API function is called as the callback routine of interrupt process MD_INTUANR corresponding to the UARTAn reception error interrupt INTUANR (process performed when the amount of data received is greater than the parameter *rxnum* specified for [UARTAn_ReceiveData](#)).

[Classification]

CG_serial_user.c

[Syntax]

```
#include "CG_macrodriver.h"
void UARTAn_SoftOverRunCallback ( UCHAR rx_data );
```

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|------------------------|---------------|
| O | UCHAR <i>rx_data</i> ; | Received data |

[Return value]

None.

UARTB n _Init

Performs initialization necessary to control the asynchronous serial interface B (UARTB) functions.

[Classification]

CG_serial.c

[Syntax]

```
void    UARTB $n$ _Init ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTB n _UserInit

Performs user-defined initialization relating to the asynchronous serial interface B (UARTB).

Remark This API function is called as the [UARTB \$n\$ _Init](#) callback routine.

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTB $n$ _UserInit ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTB n _Start

Enables asynchronous serial interface B (UARTB).

[Classification]

CG_serial.c

[Syntax]

```
void    UARTB $n$ _Start ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTB n _Stop

Disables asynchronous serial interface B (UARTB).

[Classification]

CG_serial.c

[Syntax]

```
void UARTB $n$ _Stop ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTBn_SendData

Starts UARTBn data transmission.

- Remarks 1.** This API function repeats the byte-level UARTBn transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.
- 2.** When performing a UARTBn transmission (single mode), [UARTBn_Start](#) must be called before this API function is called.
- 3.** When performing a UARTBn transmission (FIFO mode), [UARTBn_Start](#) must be called after this API function is called.

[Classification]

CG_serial.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|------------------------|---|
| I | UCHAR * <i>txbuf</i> ; | Pointer to a buffer storing the transmission data |
| I | USHORT <i>txnum</i> ; | Total amount of data to send |

[Return value]

| Macro | Description |
|---------------|---|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification - <i>txnum</i> is not a multiple of transmit FIFO as trigger. |
| MD_DATAEXISTS | Exit with error (abend) - Hold a next transmission data in the UBnTX register. |

UARTBn_ReceiveData

Starts UARTBn data reception.

- Remarks 1.** This API function performs byte-level UARTBn reception the number of times specified by the parameter *rxnum*, and stores the data in the buffer specified by the parameter *rxbuf*.
- 2.** Actual UARTBn reception starts after this API function is called, and [UARTBn_Start](#) is then called.

[Classification]

CG_serial.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|------------------------|--|
| O | UCHAR * <i>rxbuf</i> ; | Pointer to a buffer to store the received data |
| I | USHORT <i>rxnum</i> ; | Total amount of data to receive |

[Return value]

| Macro | Description |
|-------------|--|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification - <i>rxnum</i> is not a multiple of receive FIFO as trigger. |

UARTBn_SendEndCallback

Performs processing consequent to the transmission enable interrupt INTUBnTIT and the FIFO transmission completion interrupt INTUBnTIF.

Remark This API function is called as a callback routine of the interrupt process MD_INTUBnTIT corresponding to a transmission enable interrupt INTUBnTIT of UARTBn (single mode), and interrupt process MD_INTUBnTIF corresponding to a FIFO transmission completion interrupt INTUBnTIF of UARTBn (FIFO mode) (performed when total number of UARTBn transmissions specified by [UARTBn_SendData](#) parameter *txnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void UARTBn_SendEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTBn_ReceiveEndCallback

Performs processing in response to the reception completion interrupt INTUBnTIR.

Remark This API function is called as the callback routine of interrupt process MD_INTUBnTIR corresponding to the reception completion interrupt INTUBnTIR (performed when total number of UARTBn receptions specified by [UARTBn_ReceiveData](#) parameter *rxnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void UARTBn_ReceiveEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTBn_SingleErrorCallback

Performs processing in response to the reception error interrupt INTUBnTIRE (overrun error, framing error, parity error).

Remark This API function is called as a callback routine of the interrupt process MD_INTUBnTIRE corresponding to a reception error interrupt INTUBnTIRE of UARTBn (single mode).

[Classification]

CG_serial_user.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|-------------------------|--|
| O | UCHAR <i>err_type</i> ; | Trigger for reception error interrupt 00000xx1B: Overrun error 00000x1xB: Framing error 000001xxB: Parity error |

[Return value]

None.

UARTBn_FIFOErrorCallback

Performs processing in response to the reception error interrupt INTUBnTIRE (overrun error, framing error, parity error).

Remark This API function is called as a callback routine of the interrupt process MD_INTUBnTIRE corresponding to a reception error interrupt INTUBnTIRE of UARTBn (FIFO mode).

[Classification]

CG_serial_user.c

[Syntax]

```
#include "CG_macrodriver.h"
void UARTBn_FIFOErrorCallback ( UCHAR err_type1, UCHAR err_type2 );
```

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|--------------------------|--|
| O | UCHAR <i>err_type1</i> ; | Trigger for reception error interrupt 00001000B: Overrun error |
| O | UCHAR <i>err_type2</i> ; | Trigger for reception error interrupt 000000x1B: Framing error 0000001xB: Parity error |

[Return value]

None.

UARTB n _TimeoutErrorCallback

Performs processing in response to the reception timeout error interrupt INTUB n TITO.

Remark This API function is called as a callback routine of the interrupt process MD_INTUB n TITO corresponding to a reception timeout interrupt INTUBA n TITO of UARTB n (FIFO mode).

[Classification]

CG_serial_user.c

[Syntax]

```
void UARTB $n$ _TimeoutErrorCallback ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTBn_SoftOverRunCallback

Performs processing in response to detection of overrun error.

Remark This API function is called as the callback routine of interrupt process MD_INTUBnTIRE corresponding to the reception error interrupt INTUBnTIRE (process performed when the amount of data received is greater than the parameter *rxnum* specified for [UARTBn_ReceiveData](#)).

[Classification]

CG_serial_user.c

[Syntax]

```
#include "CG_macrodriver.h"
void UARTBn_SoftOverRunCallback ( UCHAR rx_data );
```

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|------------------------|---------------|
| O | UCHAR <i>rx_data</i> ; | Received data |

[Return value]

None.

UARTCn_Init

Performs initialization necessary to control the asynchronous serial interface C (UARTC) functions.

[Classification]

CG_serial.c

[Syntax]

```
void    UARTCn_Init ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTCn_UserInit

Performs user-defined initialization relating to the asynchronous serial interface C (UARTC).

Remark This API function is called as the [UARTCn_Init](#) callback routine.

[Classification]

CG_serial_user.c

[Syntax]

```
void    UARTCn_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTCn_Start

Enables asynchronous serial interface C (UARTC).

[Classification]

CG_serial.c

[Syntax]

```
void    UARTCn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTCn_Stop

Disables asynchronous serial interface C (UARTC).

[Classification]

CG_serial.c

[Syntax]

```
void    UARTCn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTCn_SendData

Starts UARTCn data transmission.

- Remarks 1.** This API function repeats the byte-level UARTCn transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.
- 2.** When performing a UARTCn transmission, [UARTCn_Start](#) must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|------------------------|---|
| I | UCHAR * <i>txbuf</i> ; | Pointer to a buffer storing the transmission data |
| I | USHORT <i>txnum</i> ; | Total amount of data to send |

[Return value]

| Macro | Description |
|---------------|---|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |
| MD_DATAEXISTS | Exit with error (abend) - Hold a next transmission data in the UCnTX register. |

UARTCn_ReceiveData

Starts UARTCn data reception.

- Remarks 1.** This API function performs byte-level UARTCn reception the number of times specified by the parameter *rxnum*, and stores the data in the buffer specified by the parameter *rxbuf*.
- 2.** Actual UARTCn reception starts after this API function is called, and [UARTCn_Start](#) is then called.

[Classification]

CG_serial.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|------------------------|--|
| O | UCHAR * <i>rxbuf</i> ; | Pointer to a buffer to store the received data |
| I | USHORT <i>rxnum</i> ; | Total amount of data to receive |

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

UARTCn_SendEndCallback

Performs processing in response to the UARTCn consecutive transmission enable interrupt INTUCnT.

Remark This API function is called as the callback routine of interrupt process MD_INTUCnT corresponding to the UARTCn consecutive transmission enable interrupt INTUCnT (performed when total number of UARTCn transmissions specified by [UARTCn_SendData](#) parameter *txnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void UARTCn_SendEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTCn_ReceiveEndCallback

Performs processing in response to the UARTCn reception completion interrupt INTUCnR.

Remark This API function is called as the callback routine of interrupt process MD_INTUCnR corresponding to the UARTCn reception completion interrupt INTUCnR (performed when total number of UARTCn receptions specified by [UARTCn_ReceiveData](#) parameter *rxnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void UARTCn_ReceiveEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

UARTCn_ErrorCallback

Performs processing in response to the UARTCn reception error interrupt INTUCnR (overrun error, framing error, parity error).

Remark This API function is called as the callback routine of interrupt process MD_INTUCnR corresponding to the UARTCn reception error interrupt INTUCnR.

[Classification]

CG_serial_user.c

[Syntax]

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

Remark n is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|-------------------------|---|
| O | UCHAR <i>err_type</i> ; | Trigger for UARTCn reception error interrupt 00000xx1B: Overrun error 00000x1xB: Framing error 000001xxB: Parity error |

[Return value]

None.

UARTCn_SoftOverRunCallback

Performs processing in response to detection of overrun error.

Remark This API function is called as the callback routine of interrupt process MD_INTUCnR corresponding to the UARTCn reception error interrupt INTUCnR (process performed when the amount of data received is greater than the parameter *rxnum* specified for [UARTCn_ReceiveData](#)).

[Classification]

CG_serial_user.c

[Syntax]

```
#include "CG_macrodriver.h"
void UARTCn_SoftOverRunCallback ( UCHAR rx_data );
```

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|------------------------|---------------|
| O | UCHAR <i>rx_data</i> ; | Received data |

[Return value]

None.

CSIB n _Init

Performs initialization necessary to control the 3-wire variable-length serial I/O B (CSIB) functions.

[Classification]

CG_serial.c

[Syntax]

```
void CSIB $n$ _Init ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIB n _UserInit

Performs user-defined initialization relating to the 3-wire variable-length serial I/O B (CSIB).

Remark This API function is called as the [CSIB \$n\$ _Init](#) callback routine.

[Classification]

CG_serial_user.c

[Syntax]

```
void CSIB $n$ _UserInit ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIB n _Start

Enables 3-wire variable-length serial I/O B (CSIB).

[Classification]

CG_serial.c

[Syntax]

```
void CSIB $n$ _Start ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIB n _Stop

Disables 3-wire variable-length serial I/O B (CSIB).

[Classification]

CG_serial.c

[Syntax]

```
void CSIB $n$ _Stop ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIBn_SendData

Starts CSIBn data transmission.

- Remarks 1.** This API function repeats the byte-level CSIBn transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.
- 2.** When performing a CSIBn transmission, [CSIBn_Start](#) must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|------------------------|---|
| I | UCHAR * <i>txbuf</i> ; | Pointer to a buffer storing the transmission data |
| I | USHORT <i>txnum</i> ; | Total amount of data to send |

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

CSIBn_ReceiveData

Starts CSIB n data reception.

- Remarks 1.** This API function performs byte-level CSIB 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 CSIB n reception, [CSIBn_Start](#) must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

```
#include "CG_macrodriver.h"
MD_STATUS CSIBn_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 |

CSIBn_SendReceiveData

Starts CSIBn data transmission/reception.

- Remarks 1.** This API function repeats the byte-level CSIBn transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.
- 2.** This API function performs byte-level CSIBn 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 CSIBn transmission/reception, [CSIBn_Start](#) must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

CSIB n _SendEndCallback

Performs processing in response to the CSIB n reception completion interrupt INTCB n R or the CSIB n consecutive transmission write enable interrupt INTCB n T.

Remark This API function is called as the callback routine (process performed when the total number of CSIB n transmissions specified in the parameter *txnum* for [CSIB \$n\$ _SendData](#) or [CSIB \$n\$ _SendReceiveData](#) has been completed) of interrupt process MD_INTCB n R corresponding to the CSIB n reception completion interrupt INTCB n R, and interrupt process MD_INTCB n T corresponding to the CSIB n consecutive transmission write enable interrupt INTCB n T.

[Classification]

CG_serial_user.c

[Syntax]

```
void CSIB $n$ _SendEndCallback ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIB n _ReceiveEndCallback

Performs processing in response to the CSIB n reception completion interrupt INTCB n R.

Remark This API function is called as the callback routine (process performed when the total number of CSIB n receptions specified in the parameter *rxnum* for [CSIB \$n\$ _ReceiveData](#) or [CSIB \$n\$ _SendReceiveData](#) has been completed) of interrupt process MD_INTCB n R corresponding to the CSIB n reception completion interrupt INTCB n R.

[Classification]

CG_serial_user.c

[Syntax]

```
void CSIB $n$ _ReceiveEndCallback ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIB n _ErrorCallback

Performs processing in response to the CSIB n reception error interrupt INTCB n R (overrun error).

Remark This API function is called as the callback routine of interrupt process MD_INTCB n R corresponding to the CSIB n reception error interrupt INTCB n R.

[Classification]

CG_serial_user.c

[Syntax]

```
void CSIB $n$ _ErrorCallback ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIE n _Init

Performs initialization necessary to control the 3-wire variable-length serial I/O E (CSIE) functions.

[Classification]

CG_serial.c

[Syntax]

```
void CSIE $n$ _Init ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIE n _UserInit

Performs user-defined initialization relating to the 3-wire variable-length serial I/O E (CSIE).

Remark This API function is called as the [CSIE \$n\$ _Init](#) callback routine.

[Classification]

CG_serial_user.c

[Syntax]

```
void CSIE $n$ _UserInit ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIE n _Start

Enables 3-wire variable-length serial I/O E (CSIE).

[Classification]

CG_serial.c

[Syntax]

```
void CSIE $n$ _Start ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIE n _Stop

Disables 3-wire variable-length serial I/O E (CSIE).

[Classification]

CG_serial.c

[Syntax]

```
void CSIE $n$ _Stop ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIE_n_SendData

Starts CSIE_n data transmission.

- Remarks 1.** This API function repeats the byte-level CSIE_n transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.
- 2.** When performing a CSIE_n transmission, [CSIE_n_Start](#) must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|------------------------|---|
| I | UCHAR * <i>txbuf</i> ; | Pointer to a buffer storing the transmission data |
| I | USHORT <i>txnum</i> ; | Total amount of data to send |

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

CSIE_n_ReceiveData

Starts CSIE_n data reception.

- Remarks 1.** This API function performs byte-level CSIE_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 CSIE_n reception, [CSIE_n_Start](#) must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|------------------------|--|
| O | UCHAR * <i>rxbuf</i> ; | Pointer to a buffer to store the received data |
| I | USHORT <i>rxnum</i> ; | Total amount of data to receive |

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

CSIE_n_SendReceiveData

Starts CSIE_n data transmission/reception.

- Remarks 1.** This API function repeats the byte-level CSIE_n transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.
- 2.** This API function performs byte-level CSIE_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 CSIE_n transmission/reception, [CSIE_n_Start](#) must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

CSIE_n_SendEndCallback

Performs processing in response to the CSIE_n transmission/reception completion interrupt INTCE_nT.

Remark This API function is called as the callback routine (process performed when the total number of CSIE_n transmissions specified in the parameter *txnum* for [CSIE_n_SendData](#) or [CSIE_n_SendReceiveData](#) has been completed) of interrupt process MD_INTCE_nT corresponding to the CSIE_n transmission/reception completion interrupt INTCE_nT.

[Classification]

CG_serial_user.c

[Syntax]

```
void CSIEn_SendEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIE_n_ReceiveEndCallback

Performs processing in response to the CSIE_n transmission/reception completion interrupt INTCE_nT.

Remark This API function is called as the callback routine (process performed when the total number of CSIE_n receptions specified in the parameter *rxnum* for [CSIE_n_ReceiveData](#) or [CSIE_n_SendReceiveData](#) has been completed) of interrupt process MD_INTCE_nT corresponding to the CSIE_n transmission/reception completion interrupt INTCE_nT.

[Classification]

CG_serial_user.c

[Syntax]

```
void CSIEn_ReceiveEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIE n _ErrorCallback

Performs processing in response to the CSIE n BUF overflow interrupt INTCE n TIOF.

Remark This API function is called as the callback routine of interrupt process MD_INTCE n TIOF corresponding to the CSIE n BUF overflow interrupt INTCE n TIOF.

[Classification]

CG_serial_user.c

[Syntax]

```
void CSIE $n$ _ErrorCallback ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIF n _Init

Performs initialization necessary to control the 3-wire variable-length serial I/O F (CSIF) functions.

[Classification]

CG_serial.c

[Syntax]

```
void CSIF $n$ _Init ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIFn_UserInit

Performs user-defined initialization relating to the 3-wire variable-length serial I/O F (CSIF).

Remark This API function is called as the [CSIFn_Init](#) callback routine.

[Classification]

CG_serial_user.c

[Syntax]

```
void CSIFn_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIF n _Start

Enables 3-wire variable-length serial I/O F (CSIF).

[Classification]

CG_serial.c

[Syntax]

```
void CSIF $n$ _Start ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIF n _Stop

Disables 3-wire variable-length serial I/O F (CSIF).

[Classification]

CG_serial.c

[Syntax]

```
void CSIF $n$ _Stop ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIFn_SendData

Starts CSIFn data transmission.

- Remarks 1.** This API function repeats the byte-level CSIFn transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.
- 2.** When performing a CSIFn transmission, [CSIFn_Start](#) must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|------------------------|---|
| I | UCHAR * <i>txbuf</i> ; | Pointer to a buffer storing the transmission data |
| I | USHORT <i>txnum</i> ; | Total amount of data to send |

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

CSIFn_ReceiveData

Starts CSIFn data reception.

- Remarks 1.** This API function performs byte-level CSIFn 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 CSIFn reception, [CSIFn_Start](#) must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|------------------------|--|
| O | UCHAR * <i>rxbuf</i> ; | Pointer to a buffer to store the received data |
| I | USHORT <i>rxnum</i> ; | Total amount of data to receive |

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

CSIFn_SendReceiveData

Starts CSIFn data transmission/reception.

- Remarks 1.** This API function repeats the byte-level CSIFn transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.
- 2.** This API function performs byte-level CSIFn 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 CSIFn transmission/reception, [CSIFn_Start](#) must be called before this API function is called.

[Classification]

CG_serial.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

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

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

CSIF n _SendEndCallback

Performs processing in response to the CSIF n transmission/reception completion interrupt INTCF n T.

Remark This API function is called as the callback routine (process performed when the total number of CSIF n transmissions specified in the parameter *txnum* for [CSIF \$n\$ _SendData](#) or [CSIF \$n\$ _SendReceiveData](#) has been completed) of interrupt process MD_INTCF n T corresponding to the CSIF n transmission/reception completion interrupt INTCF n T.

[Classification]

CG_serial_user.c

[Syntax]

```
void CSIF $n$ _SendEndCallback ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIF n _ReceiveEndCallback

Performs processing in response to the CSIF n transmission/reception completion interrupt INTCF n T.

Remark This API function is called as the callback routine (process performed when the total number of CSIF n receptions specified in the parameter *rxnum* for [CSIF \$n\$ _ReceiveData](#) or [CSIF \$n\$ _SendReceiveData](#) has been completed) of interrupt process MD_INTCF n T corresponding to the CSIF n transmission/reception completion interrupt INTCF n T.

[Classification]

CG_serial_user.c

[Syntax]

```
void CSIF $n$ _ReceiveEndCallback ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

CSIF n _ErrorCallback

Performs processing in response to the CSIF n reception error interrupt INTCF n R (overrun error).

Remark This API function is called as the callback routine of interrupt process MD_INTCF n R corresponding to the CSIF n reception error interrupt INTCF n R.

[Classification]

CG_serial_user.c

[Syntax]

```
void CSIF $n$ _ErrorCallback ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC0*n*_Init

Performs initialization necessary to control the IIC bus functions.

[Classification]

CG_serial.c

[Syntax]

```
void IIC0n_Init ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC0n_UserInit

Performs user-defined initialization relating to the IIC bus.

Remark This API function is called as the [IIC0n_Init](#) callback routine.

[Classification]

CG_serial_user.c

[Syntax]

```
void IIC0n_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC0*n*_Stop

Ends IIC0*n* communication.

[Classification]

CG_serial.c

[Syntax]

```
void IIC0n_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC0*n*_StopCondition

Generates a stop condition.

[Classification]

CG_serial.c

[Syntax]

```
void IIC0n_StopCondition ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC0n_MasterSendStart

Starts IIC0n master transmission.

Remark This API function repeats the byte-level IIC0n master transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.

[Classification]

CG_serial.c

[Syntax]

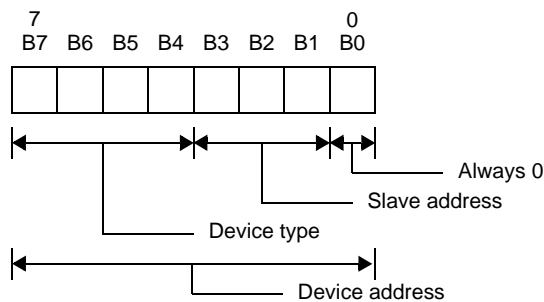
```
#include "CG_macrodriver.h"
MD_STATUS IIC0n_MasterSendStart ( UCHAR adr, UCHAR *txbuf, USHORT txnum, UCHAR wait );
```

Remark *n* is the channel number.

[Argument(s)]

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

Remark Device address *adr* consists of a device type and slave address.



[Return value]

| Macro | Description |
|----------|-------------------------|
| MD_OK | Normal completion |
| MD_ERROR | Exit with error (abend) |

IIC0n_MasterReceiveStart

Starts IIC0n master reception.

Remark This API function performs byte-level simple IIC0n 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]

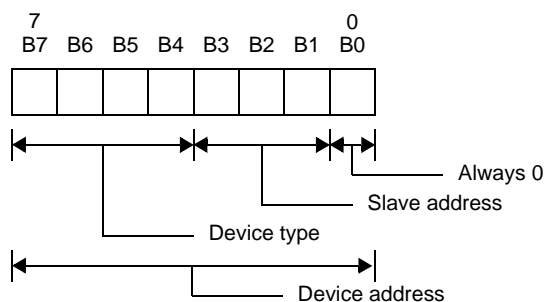
```
#include "CG_macrodriver.h"
MD_STATUS IIC0n_MasterReceiveStart ( UCHAR adr, UCHAR *rxbuf, USHORT rxnum, UCHAR wait );
```

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|-----------------------|--|
| I | UCHAR <i>adr</i> ; | Device address |
| O | UCHAR <i>*rxbuf</i> ; | Pointer to a buffer to store the received data |
| I | USHORT <i>rxnum</i> ; | Total amount of data to receive |
| I | UCHAR <i>wait</i> ; | Setup time of start conditions |

Remark Device address *adr* consists of a device type and slave address.



[Return value]

| Macro | Description |
|----------|-------------------------|
| MD_OK | Normal completion |
| MD_ERROR | Exit with error (abend) |

IIC0n_MasterSendEndCallback

Performs processing in response to the IIC0n master transfer completion interrupt INTIICn.

Remark This API function is called as the callback routine of interrupt process MD_INTIICn corresponding to the simple IIC0n master transfer completion interrupt INTIICn (performed when total number of simple IIC0n master transmissions specified by [IIC0n_MasterSendStart](#) parameter *rxnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void IIC0n_MasterSendEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC0n_MasterReceiveEndCallback

Performs processing in response to the IIC0n master transfer completion interrupt INTIICn.

Remark This API function is called as the callback routine of interrupt process MD_INTIICn corresponding to the simple IIC0n master transfer completion interrupt INTIICn (performed when total number of simple IIC0n master transmissions specified by IIC0n_MasterReceiveStart parameter *rxnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void IIC0n_MasterReceiveEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC0n_MasterErrorCallback

Performs processing in response to detection of error in IIC0n master communication.

Remark This API function is called as the callback routine (process carried out when an IIC0n master communication error is detected) of interrupt process MD_INTIICn corresponding to the IIC0n master transfer complete interrupt INTIICn.

[Classification]

CG_serial_user.c

[Syntax]

```
#include "CG_macrodriver.h"
void IIC0n_MasterErrorCallback ( MD_STATUS flag );
```

Remark n is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|-----------------|---|
| O | MD_STATUS flag; | Cause of IIC0n master communication error MD_SPT: Stop condition detected MD_NACK: Acknowledge detected |

[Return value]

None.

IIC0n_SlaveSendStart

Starts IIC0n slave transmission.

Remark This API function repeats the byte-level IIC0n 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 IIC0n_SlaveSendStart ( UCHAR *txbuf, USHORT txnum );
```

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|------------------------|---|
| I | UCHAR * <i>txbuf</i> ; | Pointer to a buffer storing the transmission data |
| I | USHORT <i>txnum</i> ; | Total amount of data to send |

[Return value]

None.

IIC0n_SlaveReceiveStart

Starts IIC0n slave reception.

Remark This API function performs byte-level IIC0n 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 IIC0n_SlaveReceiveStart ( UCHAR *rxbuf, USHORT rxnum );
```

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|------------------------|--|
| O | UCHAR * <i>rxbuf</i> ; | Pointer to a buffer to store the received data |
| I | USHORT <i>rxnum</i> ; | Total amount of data to receive |

[Return value]

None.

IIC0n_SlaveSendEndCallback

Performs processing in response to the IIC0n slave transfer completion interrupt INTIICn.

Remark This API function is called as the callback routine of interrupt process MD_INTIICn corresponding to the simple IIC0n slave transfer completion interrupt INTIICn (performed when total number of simple IIC0n slave transmissions specified by [IIC0n_SlaveSendStart](#) parameter *txnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void IIC0n_SlaveSendEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC0n_SlaveReceiveEndCallback

Performs processing in response to the IIC0n slave transfer completion interrupt INTIICn.

Remark This API function is called as the callback routine of interrupt process MD_INTIICn corresponding to the simple IIC0n slave transfer completion interrupt INTIICn (performed when total number of simple IIC0n slave transmissions specified by [IIC0n_SlaveReceiveStart](#) parameter *rxnum* has been completed).

[Classification]

CG_serial_user.c

[Syntax]

```
void IIC0n_SlaveReceiveEndCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

IIC0n_SlaveErrorCallback

Performs processing in response to detection of error in IIC0n slave communication.

Remark This API function is called as the callback routine (process carried out when an IIC0n slave communication error is detected) of interrupt process MD_INTIICn corresponding to the IIC0n slave transfer complete interrupt INTIICn.

[Classification]

CG_serial_user.c

[Syntax]

```
#include "CG_macrodriver.h"
void IIC0n_SlaveErrorCallback ( MD_STATUS flag );
```

Remark n is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|-----------------|---|
| O | MD_STATUS flag; | Cause of IIC0n master communication error MD_ERROR: Address mismatch detected MD_NACK: Acknowledge detected |

[Return value]

None.

IIC0n_GetStopConditionCallback

Performs processing in response to detection of stop condition.

Remark This API function is called as the callback routine (process carried out when a stop condition is detected) of interrupt process MD_INTIICn corresponding to the IIC0n transfer completion interrupt INTIICn.

[Classification]

CG_serial_user.c

[Syntax]

```
void IIC0n_GetStopConditionCallback ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

C.3.6 A/D Converter

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

Table C-7. 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_Start | Starts A/D conversion. |
| AD_Stop | Ends A/D conversion. |
| AD_SelectADChannel | Configures the analog voltage input pin for A/D conversion. |
| AD_SetPFTCondition | Sets the information for operation in power-fail compare mode (comparison value and A/D conversion end interrupt INTAD trigger). |
| 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_Start

Starts A/D conversion.

Remark A/D conversion is performed repeatedly between the call to this API function and a call to [AD_Stop](#).

[Classification]

CG_ad.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

[Example]

The example below shows A/D conversion of analog voltage.

[CG_main.c]

```
#include "CG_macrodriver.h"
BOOL gFlag; /* A/D conversion complete flag */
void main ( void ) {
    USHORT buffer = 0;
    int wait = 100;
    gFlag = 1; /* Initialize A/D conversion complete flag */
    .....
    AD_Start (); /* Start A/D conversion */
    while ( gFlag ); /* Wait for INTAD */
    AD_Read ( &buffer ); /* Read results of A/D conversion */
    AD_Stop (); /* End A/D conversion */
    .....
}
```

[CG_ad_user.c]

```
#include "CG_macrodriver.h"
extern BOOL gFlag; /* A/D conversion complete flag */
__interrupt void MD_INTAD ( void ) { /* Interrupt processing for INTAD */
    gFlag = 0; /* Set A/D conversion complete flag */
}
```

AD_Stop

Ends A/D conversion.

[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.

[Classification]

CG_ad.c

[Syntax]

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

[Argument(s)]

| I/O | Argument | Description |
|-----|---------------------------------|--|
| I | enum ADChannel <i>channel</i> ; | Analog voltage input pin ADCHANNEL <i>n</i> : Input pin |

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_SetPFTCondition

Sets the information for operation in power-fail compare mode (comparison value and A/D conversion end interrupt INTAD trigger).

[Classification]

CG_ad.c

[Syntax]

```
#include "CG_macrodriver.h"
#include "CG_ad.h"
MD_STATUS AD_SetPFTCondition ( UCHAR pftvalue, enum ADPFTMode mode );
```

[Argument(s)]

| I/O | Argument | Description |
|-----|------------------------------|--|
| I | UCHAR <i>pftvalue</i> ; | Comparison value |
| I | enum ADPFTMode <i>mode</i> ; | Cause of A/D conversion end interrupt INTAD EACHEND: Generate INTAD when A/D is complete PFTHIGHER: Generate INTAD if ADA0CRnH ≥ ADA0PFT PFTLOWER: Generate INTAD if ADA0CRnH < ADA0PFT |

Remark If the parameter *mode* is set to PFTHIGHER or PFTLOWER, then the value set in parameter *pftvalue* is set in the power-fail compare threshold value register (ADA0PFT), and used for comparison with the A/D conversion result register *nH* (ADA0CRnH).

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

AD_Read

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

[Classification]

CG_ad.c

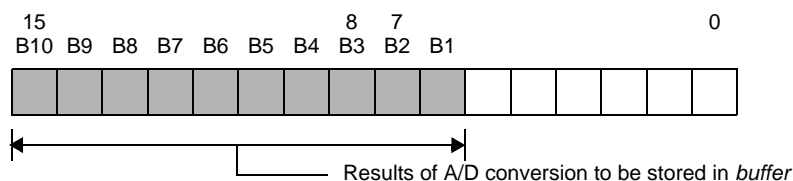
[Syntax]

```
#include "CG_macrodriver.h"
MD_STATUS 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 (10 bits) |

Remark Below is an example of the results of A/D conversion to be stored in *buffer*.



[Return value]

| Macro | Description |
|----------|-------------------------|
| MD_OK | Normal completion |
| MD_ERROR | Exit with error (abend) |

AD_ReadByte

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

[Classification]

CG_ad.c

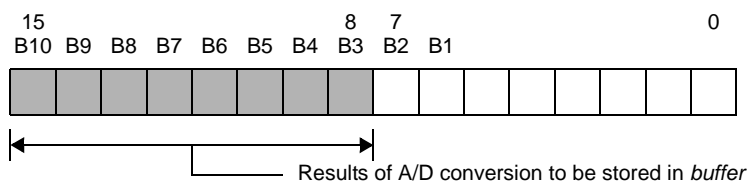
[Syntax]

```
#include "CG_macrodriver.h"
MD_STATUS 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 (8 bits; most significant 8 bits of 10-bit resolution) |

Remark Below is an example of the results of A/D conversion to be stored in *buffer*.



[Return value]

| Macro | Description |
|----------|-------------------------|
| MD_OK | Normal completion |
| MD_ERROR | Exit with error (abend) |

C.3.7 D/A Converter

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

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

| API Function Name | Function |
|------------------------------|---|
| DAn_Init | Performs initialization necessary to control D/A converter functions. |
| DAn_UserInit | Performs user-defined initialization relating to the D/A converter. |
| DAn_Start | Starts D/A conversion. |
| DAn_Stop | Ends D/A conversion. |
| DAn_SetValue | Sets the analog voltage output to the ANOn pin. |

DAn_Init

Performs initialization necessary to control D/A converter functions.

[Classification]

CG_da.c

[Syntax]

```
void DAn_Init ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

DAn_UserInit

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

Remark This API function is called as the [DAn_Init](#) callback routine.

[Classification]

CG_da_user.c

[Syntax]

```
void DAn_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

DAn_Start

Starts D/A conversion.

[Classification]

CG_da.c

[Syntax]

```
void DAn_Start ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

DAn_Stop

Ends D/A conversion.

[Classification]

CG_da.c

[Syntax]

```
void DAn_Stop ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

DAn_SetValue

Sets the analog voltage output to the ANOn pin.

[Classification]

CG_da.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|----------------------|------------------------------|
| I | UCHAR <i>value</i> ; | Analog voltage (0x0 to 0xff) |

[Return value]

None.

[Example]

Below is an example of setting "analog voltage" to channels 0 and 1

[CG_main.c]

```
#include "CG_macrodriver.h"
void main ( void ) {
    .....
    DA0_Start (); /* Start D/A conversion */
    DA1_Start (); /* Start D/A conversion */
    .....
    DA0_SetValue ( 0x7f ); /* Set analog voltage */
    .....
}
```

[CG_tau_user.c]

```
#include "CG_macrodriver.h"
UCHAR gValue = 0;
__interrupt void MD_INTTM05 ( void ) { /* Interrupt processing for INTTM05 */
    DA1_SetValue ( gValue++ ); /* Set analog voltage */
}
```

C.3.8 Timer

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

Table C-9. API Functions: [Timer]

| API Function Name | Function |
|----------------------------|---|
| TMPn_Init | Performs initialization necessary to control 16-bit timer/event counter P (TMP) functions. |
| TMPn_UserInit | Performs user-defined initialization relating to the 16-bit timer/event counter P (TMP). |
| TMPn_Start | Starts the count for 16-bit timer/event counter P (TMP). |
| TMPn_Stop | Ends the count for 16-bit timer/event counter P (TMP). |
| TMPn_ChangeTimerCondition | Changes the counter value for 16-bit timer/event counter P (TMP). |
| TMPn_GetPulseWidth | Reads the pulse width of 16-bit timer/event counter P (TMP) (high/low level width). |
| TMPn_GetFreeRunningValue | Reads the value captured by 16-bit timer/event counter P (TMP). |
| TMPn_ChangeDuty | Changes the duty ratio of the PWM signal. |
| TMPn_SoftwareTriggerOn | Generates the trigger (software trigger) for timer output. |
| TMQ0_Init | Performs initialization necessary to control 16-bit timer/event counter Q (TMQ) functions. |
| TMQ0_UserInit | Performs user-defined initialization relating to the 16-bit timer/event counter Q (TMQ). |
| TMQ0_Start | Starts the count for 16-bit timer/event counter Q (TMQ). |
| TMQ0_Stop | Ends the count for 16-bit timer/event counter Q (TMQ). |
| TMQ0_ChangeTimerCondition | Changes the counter value for 16-bit timer/event counter Q (TMQ). |
| TMQ0_GetPulseWidth | Reads the pulse width of 16-bit timer/event counter Q (TMQ) (high/low level width). |
| TMQ0_GetFreeRunningValue | Reads the value captured by 16-bit timer/event counter Q (TMQ). |
| TMQ0_ChangeDuty | Changes the duty ratio of the PWM signal. |
| TMQ0_SoftwareTriggerOn | Generates the trigger (software trigger) for timer output. |
| TAAAn_Init | Performs initialization necessary to control 16-bit timer/event counter AA (TAA) functions. |
| TAAAn_UserInit | Performs user-defined initialization relating to the 16-bit timer/event counter AA (TAA). |
| TAAAn_Start | Starts the count for 16-bit timer/event counter AA (TAA). |
| TAAAn_Stop | Ends the count for 16-bit timer/event counter AA (TAA). |
| TAAAn_ChangeTimerCondition | Changes the counter value for 16-bit timer/event counter AA (TAA). |
| TAAAn_ControlOutputToggle | Changes the toggle control of 16-bit timer/event counter AA (TAA). |
| TAAAn_GetPulseWidth | Reads the pulse width of 16-bit timer/event counter AA (TAA) (high/low level width). |
| TAAAn_GetFreeRunningValue | Reads the value captured by 16-bit timer/event counter AA (TAA). |
| TAAAn_ChangeDuty | Changes the duty ratio of the PWM signal. |
| TAAAn_SoftwareTriggerOn | Generates the trigger (software trigger) for timer output. |
| TABn_Init | Performs initialization necessary to control 16-bit timer/event counter AB (TAB) functions. |
| TABn_UserInit | Performs user-defined initialization relating to the 16-bit timer/event counter AB (TAB). |
| TABn_Start | Starts the count for 16-bit timer/event counter AB (TAB). |

| API Function Name | Function |
|---------------------------|---|
| TABn_Stop | Ends the count for 16-bit timer/event counter AB (TAB). |
| TABn_ChangeTimerCondition | Changes the counter value for 16-bit timer/event counter AB (TAB). |
| TABn_ControlOutputToggle | Changes the toggle control of 16-bit timer/event counter AB (TAB). |
| TABn_GetPulseWidth | Reads the pulse width of 16-bit timer/event counter AB (TAB) (high/low level width). |
| TABn_GetFreeRunningValue | Reads the value captured by 16-bit timer/event counter AB (TAB). |
| TABn_ChangeDuty | Changes the duty ratio of the PWM signal. |
| TABn_SoftwareTriggerOn | Generates the trigger (software trigger) for timer output. |
| TMT0_Init | Performs initialization necessary to control 16-bit timer/event counter T (TMT) functions. |
| TMT0_UserInit | Performs user-defined initialization relating to the 16-bit timer/event counter T (TMT). |
| TMT0_Start | Starts the count for 16-bit timer/event counter T (TMT). |
| TMT0_Stop | Ends the count for 16-bit timer/event counter T (TMT). |
| TMT0_ChangeTimerCondition | Changes the counter value for 16-bit timer/event counter T (TMT). |
| TMT0_GetPulseWidth | Reads the pulse width of 16-bit timer/event counter T (TMT) (high/low level width). |
| TMT0_GetFreeRunningValue | Reads the value captured by 16-bit timer/event counter T (TMT). |
| TMT0_ChangeDuty | Changes the duty ratio of the PWM signal. |
| TMT0_SoftwareTriggerOn | Generates the trigger (software trigger) for timer output. |
| TMT0_EnableHold | Changes the encoder counter control of 16-bit timer/event counter T (TMT) to holding count value. |
| TMT0_DisableHold | Changes the encoder counter control of 16-bit timer/event counter T (TMT) to normal operation. |
| TMT0_ChangeCountValue | Changes the initial counter value for 16-bit timer/event counter T (TMT). |
| TMMn_Init | Performs initialization necessary to control 16-bit interval timer M (TMM) functions. |
| TMMn_UserInit | Performs user-defined initialization relating to the 16-bit interval timer M (TMM). |
| TMMn_Start | Starts the count for 16-bit interval timer M (TMM). |
| TMMn_Stop | Ends the count for 16-bit interval timer M (TMM). |
| TMMn_ChangeTimerCondition | Changes the counter value for 16-bit interval timer M (TMM). |

TMP n _Init

Performs initialization necessary to control 16-bit timer/event counter P (TMP) functions.

[Classification]

CG_timer.c

[Syntax]

```
void    TMPn_Init ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMP n _UserInit

Performs user-defined initialization relating to the 16-bit timer/event counter P (TMP).

Remark This API function is called as the [TMP \$n\$ _Init](#) callback routine.

[Classification]

CG_timer_user.c

[Syntax]

```
void    TMP $n$ _UserInit ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMP n _Start

Starts the count for 16-bit timer/event counter P (TMP).

Remark The length of time between the call to this API function and the start of counting will vary depending on the function type (e.g. interval timer, external event counter, or external trigger pulse output).

[Classification]

CG_timer.c

[Syntax]

```
void    TMPn_Start ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMP n _Stop

Ends the count for 16-bit timer/event counter P (TMP).

Remark The length of time between the call to this API function and the end of counting will vary depending on the function type (e.g. interval timer, external event counter, or external trigger pulse output).

[Classification]

CG_timer.c

[Syntax]

```
void    TMPn_Stop ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMPn_ChangeTimerCondition

Changes the counter value for 16-bit timer/event counter P (TMP).

Remark The value specified in parameter *array_reg* is set in TMPn capture/compare register *m* (TPnCCR*m*).

[Classification]

CG_timer.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|--------------------|--|
| I | USHORT *array_reg; | Pointer to an area storing the count value (0x0 to 0xffff) |
| I | UCHAR array_num; | Register to change 1: TPnCCR0 2: TPnCCR0, TPnCCR1 |

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

[Example]

The example below shows changing the interval time to one half.
In this example, channel 0 has been selected for the interval timer.

[CG_main.c]

```
#include "CG_macrodriver.h"
void main ( void ) {
    int flag_finish = 1;
    USHORT array_reg = TMP_TP0CCR0_VALUE >> 1; /* TMP_TP0CCR0_VALUE: Current
interval time */
    UCHAR array_num = 1;
    .....
    TMP0_Start (); /* Start count */
    while ( flag_finish ); /* Check for time up */
```

```
.....  
    TMP0_ChanneTimerCondition ( &array_reg, array_num );    /* Change counter value */  
.....  
}
```

TMPn_GetPulseWidth

Reads the pulse width of 16-bit timer/event counter P (TMP) (high/low level width).

- Remarks 1.** This API function can only be called when 16-bit timer/event counter P (TMP) is being used for pulse width measurement.
- 2.** If there is an overflow (2 pulses or more) during pulse-width measurement, then the pulse width will not be read correctly.

[Classification]

CG_timer.c

[Syntax]

```
#include "CG_macrodriver.h"
void TMPn_GetPulseWidth ( ULONG *activewith, ULONG *inactivewidth );
```

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|-------------------------------|--|
| O | ULONG <i>*activewith</i> ; | Pointer to an area storing the high level width that was read (0x0 to 0x1ffff) |
| O | ULONG <i>*inactivewidth</i> ; | Pointer to an area storing the low level width that was read (0x0 to 0x1ffff) |

[Return value]

None.

TMP n _GetFreeRunningValue

Reads the value captured by 16-bit timer/event counter P (TMP).

Remark This API function can only be called when 16-bit timer/event counter P (TMP) is used as a free-running timer, and TMP n capture/compare register m (TP n CCR m) is selected as the capture register.

[Classification]

CG_timer.c

[Syntax]

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

Remark n is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|-------------------------|---|
| O | ULONG *count; | Pointer to area in which to store the width that was read |
| I | enum TMChannel channel; | Channel to read TMCHANNEL0: Channel 0 (TP n CCR0) TMCHANNEL1: Channel 1 (TP n CCR1) |

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

TMPn_ChangeDuty

Changes the duty ratio of the PWM signal.

Remark This API function can only be called when 16-bit timer/event counter P (TMP) is used for external trigger pulse output / PWM output.

[Classification]

CG_timer.c

[Syntax]

```
#include "CG_macrodriver.h"
void TMPn_ChangeDuty ( UCHAR array_duty );
```

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|---------------------------|--------------------------------|
| I | UCHAR <i>array_duty</i> ; | Duty ratio (0 to 100, unit: %) |

Remark The value set to duty ratio *array_duty* 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 ) {
    int flagStatus = 1;
    UCHAR array_duty = 25;
    .....
    TMP0_Start (); /* Start count */
    while ( flagStatus );
    TMP0_ChangeDuty ( array_duty ); /* Change duty ratio */
    .....
}
```

TMP n _SoftwareTriggerOn

Generates the trigger (software trigger) for timer output.

Remark This API function can only be called when 16-bit timer/event counter P (TMP) is used for external trigger pulse output / one-shot pulse output.

[Classification]

CG_timer.c

[Syntax]

```
void    TMPn_SoftwareTriggerOn ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMQ0_Init

Performs initialization necessary to control 16-bit timer/event counter Q (TMQ) functions.

[Classification]

CG_timer.c

[Syntax]

```
void    TMQ0_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

TMQ0_UserInit

Performs user-defined initialization relating to the 16-bit timer/event counter Q (TMQ).

Remark This API function is called as the [TMQ0_Init](#) callback routine.

[Classification]

CG_timer_user.c

[Syntax]

```
void    TMQ0_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

TMQ0_Start

Starts the count for 16-bit timer/event counter Q (TMQ).

Remark The length of time between the call to this API function and the start of counting will vary depending on the function type (e.g. interval timer, external event counter, or external trigger pulse output).

[Classification]

CG_timer.c

[Syntax]

```
void    TMQ0_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

TMQ0_Stop

Ends the count for 16-bit timer/event counter Q (TMQ).

Remark The length of time between the call to this API function and the start of counting will vary depending on the function type (e.g. interval timer, external event counter, or external trigger pulse output).

[Classification]

CG_timer.c

[Syntax]

```
void    TMQ0_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

TMQ0_ChangeTimerCondition

Changes the counter value for 16-bit timer/event counter Q (TMQ).

Remark The value specified in parameter *array_reg* is set to TMQ0 capture/compare register *m* "TQ0CCR*m*".

[Classification]

CG_timer.c

[Syntax]

```
#include "CG_macrodriver.h"
MD_STATUS TMQ0_ChangeTimerCondition ( USHORT *array_reg, UCHAR array_num );
```

[Argument(s)]

| I/O | Argument | Description |
|-----|--------------------|--|
| I | USHORT *array_reg; | Pointer to an area storing the count value (0x0 to 0xffff) |
| I | UCHAR array_num; | Register to change 1: TQ0CCR0 2: TQ0CCR0, TQ0CCR1 3: TQ0CCR0, TQ0CCR1, TQ0CCR2 4: TQ0CCR0, TQ0CCR1, TQ0CCR2, TQ0CCR3 |

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

[Example]

The example below shows changing the interval time to one half.
In this example, channel 0 has been selected for the interval timer.

[CG_main.c]

```
#include "CG_macrodriver.h"
void main ( void ) {
    int flag_finish = 1;
    USHORT array_reg = TMQ_TQ0CCR0_VALUE >> 1; /* TMQ_TQ0CCR0_VALUE: Current
interval time */
    UCHAR array_num = 1;
    .....
    TMQ0_Start (); /* Start count */
    while ( flag_finish ); /* Check for time up */
```

```
.....  
    TMQ0_ChanneTimerCondition ( &array_reg, array_num );    /* Change counter value */  
.....  
}
```


TMQ0_GetPulseWidth

Reads the pulse width of 16-bit timer/event counter Q (TMQ) (high/low level width).

- Remarks 1.** This API function can only be called when 16-bit timer/event counter Q (TMQ) is being used for pulse width measurement.
- 2.** If there is an overflow (2 pulses or more) during pulse-width measurement, then the pulse width will not be read correctly.

[Classification]

CG_timer.c

[Syntax]

```
#include "CG_macrodriver.h"
void TMQ0_GetPulseWidth ( ULONG *activewith, ULONG *inactivewidth );
```

[Argument(s)]

| I/O | Argument | Description |
|-----|-----------------------|--|
| O | ULONG *activewith; | Pointer to an area storing the high level width that was read (0x0 to 0x1ffff) |
| O | ULONG *inactivewidth; | Pointer to an area storing the low level width that was read (0x0 to 0x1ffff) |

[Return value]

None.

TMQ0_GetFreeRunningValue

Reads the value captured by 16-bit timer/event counter Q (TMQ).

Remark This API function can only be called when 16-bit timer/event counter Q (TMQ) is used as a free-running timer, and TQ0 capture/compare register *m* (TQ0CCR*m*) is selected as the capture register.

[Classification]

CG_timer.c

[Syntax]

```
#include "CG_macrodriver.h"
#include "CG_timer.h"
MD_STATUS TMQ0_GetFreeRunningValue ( ULONG *count, enum TMChannel channel );
```

[Argument(s)]

| I/O | Argument | Description |
|-----|-------------------------|---|
| O | ULONG *count; | Pointer to area in which to store the width that was read |
| I | enum TMChannel channel; | Channel to read TMCHANNEL0: Channel 0 (TQ0CCR0) TMCHANNEL1: Channel 1 (TQ0CCR1) TMCHANNEL2: Channel 2 (TQ0CCR2) TMCHANNEL3: Channel 3 (TQ0CCR3) |

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

TMQ0_ChangeDuty

Changes the duty ratio of the PWM signal.

Remark This API function can only be called when 16-bit timer/event counter Q (TMQ) is used for external trigger pulse output / PWM output.

[Classification]

CG_timer.c

[Syntax]

```
#include "CG_macrodriver.h"
MD_STATUS TMQ0_ChangeDuty ( UCHAR *array_duty, UCHAR array_num );
```

[Argument(s)]

| I/O | Argument | Description |
|-----|--------------------|---|
| I | UCHAR *array_duty; | Pointer to an area storing the duty ratio (0 to 100; in percent) |
| I | UCHAR array_num; | Register to change 1: TQ0CCR1 2: TQ0CCR1, TQ0CCR2 3: TQ0CCR1, TQ0CCR2, TQ0CCR3 |

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

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

[Example]

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

[CG_main.c]

```
#include "CG_macrodriver.h"
void main ( void ) {
    int flagStatus = 1;
    UCHAR array_duty = 25;
    UCHAR array_num = 1;
    .....
    TMQ0_Start (); /* Start count */
    while ( flagStatus );
```

```
TMQ0_ChangeDuty ( &array_duty, array_num );    /*Change duty ratio */  
.....  
}
```

TMQ0_SoftwareTriggerOn

Generates the trigger (software trigger) for timer output.

Remark This API function can only be called when 16-bit timer/event counter Q (TMQ) is used for external trigger pulse output / one-shot pulse output.

[Classification]

CG_timer.c

[Syntax]

```
void    TMQ0_SoftwareTriggerOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

TAA n _Init

Performs initialization necessary to control 16-bit timer/event counter AA (TAA) functions.

[Classification]

CG_timer.c

[Syntax]

```
void TAA $n$ _Init ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

TAA n _UserInit

Performs user-defined initialization relating to the 16-bit timer/event counter AA (TAA).

Remark This API function is called as the [TAA \$n\$ _Init](#) callback routine.

[Classification]

CG_timer_user.c

[Syntax]

```
void TAA $n$ _UserInit ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

TAA n _Start

Starts the count for 16-bit timer/event counter AA (TAA).

Remark The length of time between the call to this API function and the start of counting will vary depending on the function type (e.g. interval timer, external event counter, or external trigger pulse output).

[Classification]

CG_timer.c

[Syntax]

```
void TAA $n$ _Start ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

TAA n _Stop

Ends the count for 16-bit timer/event counter AA (TAA).

Remark The length of time between the call to this API function and the end of counting will vary depending on the function type (e.g. interval timer, external event counter, or external trigger pulse output).

[Classification]

CG_timer.c

[Syntax]

```
void TAA $n$ _Stop ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

TAA_n_ChangeTimerCondition

Changes the counter value for 16-bit timer/event counter AA (TAA).

Remark The value specified in parameter *arrar_reg* is set in TAA_n capture/compare register *m* (TAA_nCCR*m*).

[Classification]

CG_timer.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|--------------------|---|
| I | USHORT *array_reg; | Pointer to an area storing the count value (0x0 to 0xffff) |
| I | UCHAR array_num; | Register to change 1: TAA _n CCR0 2: TAA _n CCR0, TAA _n CCR1 |

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

TAA n _ControlOutputToggle

Changes the toggle control of 16-bit timer/event counter AA (TAA).

Remark This API function can only be called when 16-bit timer/event counter AA (TAA) is used for interval timer / free-running timer.

[Classification]

CG_timer.c

[Syntax]

```
#include "CG_macrodriver.h"
#include "CG_timer.h"
MD_STATUS TAA $n$ _ControlOutputToggle ( enum TMO $u$ t toggle, enum TMChannel channel );
```

Remark n is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|---------------------------------|---|
| I | enum TMO u t <i>toggle</i> ; | Toggle control STANDARD: Normal toggle operation INACTIVE: Reset request ACTIVE: Set request FREEZE: Keep request |
| I | enum TMChannel <i>channel</i> ; | pin to change TMCHANNEL0: TOAA n 0 pin TMCHANNEL1: TOAA n 1 pin |

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

TAA_n_GetPulseWidth

Reads the pulse width of 16-bit timer/event counter AA (TAA) (high/low level width).

- Remarks 1.** This API function can only be called when 16-bit timer/event counter AA (TAA) is being used for pulse width measurement.
- 2.** If there is an overflow (2 pulses or more) during pulse-width measurement, then the pulse width will not be read correctly.

[Classification]

CG_timer.c

[Syntax]

```
#include "CG_macrodriver.h"
void TAAn_GetPulseWidth ( ULONG *activewith, ULONG *inactivewidth );
```

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|-----------------------|--|
| O | ULONG *activewith; | Pointer to an area storing the high level width that was read (0x0 to 0x1ffff) |
| O | ULONG *inactivewidth; | Pointer to an area storing the low level width that was read (0x0 to 0x1ffff) |

[Return value]

None.

TAA n _GetFreeRunningValue

Reads the value captured by 16-bit timer/event counter AA (TAA).

Remark This API function can only be called when 16-bit timer/event counter AA (TAA) is used as a free-running timer, and TAA n capture/compare register m (TAA n CCR m) is selected as the capture register.

[Classification]

CG_timer.c

[Syntax]

```
#include "CG_macrodriver.h"
#include "CG_timer.h"
MD_STATUS TAA $n$ _GetFreeRunningValue ( ULONG *count, enum TMChannel channel );
```

Remark n is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|-------------------------|---|
| O | ULONG *count; | Pointer to area in which to store the width that was read |
| I | enum TMChannel channel; | Channel to read TMCHANNEL0: Channel 0 (TAA n CCR0) TMCHANNEL1: Channel 1 (TAA n CCR1) |

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

TAA n _ChangeDuty

Changes the duty ratio of the PWM signal.

Remark This API function can only be called when 16-bit timer/event counter AA (TAA) is used for external trigger pulse output / PWM output.

[Classification]

CG_timer.c

[Syntax]

```
#include "CG_macrodriver.h"
void TAA $n$ _ChangeDuty ( UCHAR array_duty );
```

Remark n is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|---------------------------|--------------------------------|
| I | UCHAR <i>array_duty</i> ; | Duty ratio (0 to 100, unit: %) |

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

[Return value]

None.

TAA n _SoftwareTriggerOn

Generates the trigger (software trigger) for timer output.

Remark This API function can only be called when 16-bit timer/event counter AA (TAA) is used for external trigger pulse output / one-shot pulse output.

[Classification]

CG_timer.c

[Syntax]

```
void TAA $n$ _SoftwareTriggerOn ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

TAB n _Init

Performs initialization necessary to control 16-bit timer/event counter AB (TAB) functions.

[Classification]

CG_timer.c

[Syntax]

```
void TABn_Init ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

TAB n _UserInit

Performs user-defined initialization relating to the 16-bit timer/event counter AB (TAB).

Remark This API function is called as the [TAB \$n\$ _Init](#) callback routine.

[Classification]

CG_timer_user.c

[Syntax]

```
void TAB $n$ _UserInit ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

TAB n _Start

Starts the count for 16-bit timer/event counter AB (TAB).

Remark The length of time between the call to this API function and the start of counting will vary depending on the function type (e.g. interval timer, external event counter, or external trigger pulse output).

[Classification]

CG_timer.c

[Syntax]

```
void TAB $n$ _Start ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

TAB n _Stop

Ends the count for 16-bit timer/event counter AB (TAB).

Remark The length of time between the call to this API function and the end of counting will vary depending on the function type (e.g. interval timer, external event counter, or external trigger pulse output).

[Classification]

CG_timer.c

[Syntax]

```
void TAB $n$ _Stop ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

TAB_n_ChangeTimerCondition

Changes the counter value for 16-bit timer/event counter AB (TAB).

Remark The value specified in parameter *array_reg* is set in TAB_n capture/compare register *m* (TAB_nCCR_m).

[Classification]

CG_timer.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|--------------------|--|
| I | USHORT *array_reg; | Pointer to an area storing the count value (0x0 to 0xffff) |
| I | UCHAR array_num; | Register to change 1: TAB _n CCR0 2: TAB _n CCR0, TAB _n CCR1 3: TAB _n CCR0, TAB _n CCR1, TAB _n CCR2 4: TAB _n CCR0, TAB _n CCR1, TAB _n CCR2, TAB _n CCR3 |

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

TAB n _ControlOutputToggle

Changes the toggle control of 16-bit timer/event counter AB (TAB).

Remark This API function can only be called when 16-bit timer/event counter AB (TAB) is used for interval timer / free-running timer.

[Classification]

CG_timer.c

[Syntax]

```
#include "CG_macrodriver.h"
#include "CG_timer.h"
MD_STATUS TABn_ControlOutputToggle ( enum TMOut toggle, enum TMChannel channel );
```

Remark n is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|---------------------------------|---|
| I | enum TMOutput <i>toggle</i> ; | Toggle control STANDARD: Normal toggle operation INACTIVE: Reset request ACTIVE: Set request FREEZE: Keep request |
| I | enum TMChannel <i>channel</i> ; | pin to change TMCHANNEL0: TOAB n 0 pin TMCHANNEL1: TOAB n 1 pin |

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

TAB_n_GetPulseWidth

Reads the pulse width of 16-bit timer/event counter AB (TAB) (high/low level width).

- Remarks 1.** This API function can only be called when 16-bit timer/event counter AB (TAB) is being used for pulse width measurement.
- 2.** If there is an overflow (2 pulses or more) during pulse-width measurement, then the pulse width will not be read correctly.

[Classification]

CG_timer.c

[Syntax]

```
#include "CG_macrodriver.h"
void TABn_GetPulseWidth ( ULONG *activewith, ULONG *inactivewidth );
```

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|-----------------------|--|
| O | ULONG *activewith; | Pointer to an area storing the high level width that was read (0x0 to 0x1ffff) |
| O | ULONG *inactivewidth; | Pointer to an area storing the low level width that was read (0x0 to 0x1ffff) |

[Return value]

None.

TAB_n_GetFreeRunningValue

Reads the value captured by 16-bit timer/event counter AB (TAB).

Remark This API function can only be called when 16-bit timer/event counter AB (TAB) is used as a free-running timer, and TAB_n capture/compare register *m* (TAB_nCCR*m*) is selected as the capture register.

[Classification]

CG_timer.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|-------------------------|---|
| O | ULONG *count; | Pointer to area in which to store the width that was read |
| I | enum TMChannel channel; | Channel to read TMCHANNEL0: Channel 0 (TAB _n CCR0) TMCHANNEL1: Channel 1 (TAB _n CCR1) TMCHANNEL2: Channel 2 (TAB _n CCR2) TMCHANNEL3: Channel 3 (TAB _n CCR3) |

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

TAB n _ChangeDuty

Changes the duty ratio of the PWM signal.

Remark This API function can only be called when 16-bit timer/event counter AB (TAB) is used for external trigger pulse output / PWM output.

[Classification]

CG_timer.c

[Syntax]

```
#include "CG_macrodriver.h"
MD_ATATUS TABn_ChangeDuty ( UCHAR array_duty, UCHAR array_num );
```

Remark n is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|---------------------------|--|
| I | UCHAR <i>array_duty</i> ; | Duty ratio (0 to 100, unit: %) |
| I | UCHAR <i>array_num</i> ; | Register to change 1: TAB n CCR1 2: TAB n CCR1, TP n CCR2 3: TAB n CCR1, TP n CCR2, TP n CCR3 |

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

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

TAB n _SoftwareTriggerOn

Generates the trigger (software trigger) for timer output.

Remark This API function can only be called when 16-bit timer/event counter AB (TAB) is used for external trigger pulse output / one-shot pulse output.

[Classification]

CG_timer.c

[Syntax]

```
void TAB $n$ _SoftwareTriggerOn ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMT0_Init

Performs initialization necessary to control 16-bit timer/event counter T (TMT) functions.

[Classification]

CG_timer.c

[Syntax]

```
void TMT0_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

TMT0_UserInit

Performs user-defined initialization relating to the 16-bit timer/event counter T (TMT).

Remark This API function is called as the [TMT0_Init](#) callback routine.

[Classification]

CG_timer_user.c

[Syntax]

```
void TMT0_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

TMT0_Start

Starts the count for 16-bit timer/event counter T (TMT).

Remark The length of time between the call to this API function and the start of counting will vary depending on the function type (e.g. interval timer, external event counter, or external trigger pulse output).

[Classification]

CG_timer.c

[Syntax]

```
void TMT0_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

TMT0_Stop

Ends the count for 16-bit timer/event counter T (TMT).

Remark The length of time between the call to this API function and the end of counting will vary depending on the function type (e.g. interval timer, external event counter, or external trigger pulse output).

[Classification]

CG_timer.c

[Syntax]

```
void TMT0_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

TMT0_ChangeTimerCondition

Changes the counter value for 16-bit timer/event counter T (TMT).

Remark The value specified in parameter *arrar_reg* is set in TMT0 capture/compare register *m* (TT0CCR*m*).

[Classification]

CG_timer.c

[Syntax]

```
#include "CG_macrodriver.h"
MD_STATUS TMT0_ChangeTimerCondition ( USHORT *array_reg, UCHAR array_num );
```

[Argument(s)]

| I/O | Argument | Description |
|-----|--------------------|--|
| I | USHORT *array_reg; | Pointer to an area storing the count value (0x0 to 0xffff) |
| I | UCHAR array_num; | Register to change 1: TT0CCR0 2: TT0CCR0, TT0CCR1 |

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

TMT0_GetPulseWidth

Reads the pulse width of 16-bit timer/event counter T (TMT) (high/low level width).

- Remarks 1.** This API function can only be called when 16-bit timer/event counter T (TMT) is being used for pulse width measurement.
- 2.** If there is an overflow (2 pulses or more) during pulse-width measurement, then the pulse width will not be read correctly.

[Classification]

CG_timer.c

[Syntax]

```
#include "CG_macrodriver.h"
void TMT0_GetPulseWidth ( ULONG *activewith, ULONG *inactivewidth );
```

[Argument(s)]

| I/O | Argument | Description |
|-----|-------------------------------|--|
| O | ULONG <i>*activewith</i> ; | Pointer to an area storing the high level width that was read (0x0 to 0x1ffff) |
| O | ULONG <i>*inactivewidth</i> ; | Pointer to an area storing the low level width that was read (0x0 to 0x1ffff) |

[Return value]

None.

TMT0_GetFreeRunningValue

Reads the value captured by 16-bit timer/event counter T (TMT).

Remark This API function can only be called when 16-bit timer/event counter T (TMT) is used as a free-running timer, and TMT0 capture/compare register *m* (TT0CCR*m*) is selected as the capture register.

[Classification]

CG_timer.c

[Syntax]

```
#include "CG_macrodriver.h"
#include "CG_timer.h"
MD_STATUS TMT0_GetFreeRunningValue ( ULONG *count, enum TMChannel channel );
```

[Argument(s)]

| I/O | Argument | Description |
|-----|-------------------------|---|
| O | ULONG *count; | Pointer to area in which to store the width that was read |
| I | enum TMChannel channel; | Channel to read TMCHANNEL0: Channel 0 (TPnCCR0) TMCHANNEL1: Channel 1 (TPnCCR1) |

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

TMT0_ChangeDuty

Changes the duty ratio of the PWM signal.

Remark This API function can only be called when 16-bit timer/event counter T (TMT) is used for external trigger pulse output / PWM output.

[Classification]

CG_timer.c

[Syntax]

```
#include "CG_macrodriver.h"
MD_STATUS TMT0_ChangeDuty ( UCHAR array_duty );
```

[Argument(s)]

| I/O | Argument | Description |
|-----|---------------------------|--------------------------------|
| I | UCHAR <i>array_duty</i> ; | Duty ratio (0 to 100, unit: %) |

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

[[Return value]]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

TMT0_SoftwareTriggerOn

Generates the trigger (software trigger) for timer output.

Remark This API function can only be called when 16-bit timer/event counter T (TMT) is used for external trigger pulse output / one-shot pulse output.

[Classification]

CG_timer.c

[Syntax]

```
void TMT0_SoftwareTriggerOn ( void );
```

[Argument(s)]

None.

[Return value]

None.

TMT0_EnableHold

Changes the encoder counter control of 16-bit timer/event counter T (TMT) to holding count value.

Remark This API function can only be called when 16-bit timer/event counter T (TMT) is used for encoder count.

[Classification]

CG_timer.c

[Syntax]

```
void TMT0_EnableHold ( void );
```

[Argument(s)]

None.

[Return value]

None.

TMT0_DisableHold

Changes the encoder counter control of 16-bit timer/event counter T (TMT) to normal operation.

Remark This API function can only be called when 16-bit timer/event counter T (TMT) is used for encoder count.

[Classification]

CG_timer.c

[Syntax]

```
void TMT0_DisableHold ( void );
```

[Argument(s)]

None.

[Return value]

None.

TMT0_ChangeCountValue

Changes the initial counter value for 16-bit timer/event counter T (TMT).

- Remarks 1.** The value specified in parameter *regvalue* is set in TMT0 counter write register (TT0TCW).
- 2.** This API function can only be called when 16-bit timer/event counter T (TMT) is used for encoder count.

[Classification]

CG_timer.c

[Syntax]

```
#include "CG_macrodriver.h"
void TMT0_ChangeCountValue ( USHORT regvalue );
```

[Argument(s)]

| I/O | Argument | Description |
|-----|--------------------------|-----------------------------|
| I | USHORT <i>regvalue</i> ; | Count value (0x0 to 0xffff) |

[Return value]

None.

TMM n _Init

Performs initialization necessary to control 16-bit interval timer M (TMM) functions.

[Classification]

CG_timer.c

[Syntax]

```
void TMM $n$ _Init ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMM n _UserInit

Performs user-defined initialization relating to the 16-bit interval timer M (TMM).

Remark This API function is called as the [TMM \$n\$ _Init](#) callback routine.

[Classification]

CG_timer_user.c

[Syntax]

```
void TMM $n$ _UserInit ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMM n _Start

Starts the count for 16-bit interval timer M (TMM).

[Classification]

CG_timer.c

[Syntax]

```
void TMM $n$ _Start ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMM n _Stop

Ends the count for 16-bit interval timer M (TMM).

[Classification]

CG_timer.c

[Syntax]

```
void TMM $n$ _Stop ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

TMMn_ChangeTimerCondition

Changes the counter value for 16-bit interval timer M (TMM).

Remark The value specified in parameter *regvalue* is set to TMMn control register 0 "TMnCMP0".

[Classification]

CG_timer.c

[Syntax]

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

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|--------------------------|-------------------------------|
| I | USHORT <i>regvalue</i> ; | Counter value (0x0 to 0xffff) |

[Return value]

None.

C.3.9 Watch Timer

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

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

| API Function Name | Function |
|-------------------|---|
| WT_Init | Performs initialization necessary to control watch timer functions. |
| WT_UserInit | Performs user-defined initialization relating to the watch timer. |
| WT_Start | Clears the watch timer counter and resumes counting. |
| WT_Stop | Ends the count for watch timer. |

WT_Init

Performs initialization necessary to control watch timer functions.

[Classification]

CG_wt.c

[Syntax]

```
void WT_Init ( void );
```

[Argument(s)]

None.

[Return value]

None.

WT_UserInit

Performs user-defined initialization relating to the watch timer.

Remark This API function is called as the [WT_Init](#) callback routine.

[Classification]

CG_wt_user.c

[Syntax]

```
void WT_UserInit ( void );
```

[Argument(s)]

None.

[Return value]

None.

WT_Start

Clears the watch timer counter and resumes counting.

[Classification]

CG_wt.c

[Syntax]

```
void WT_Start ( void );
```

[Argument(s)]

None.

[Return value]

None.

WT_Stop

Ends the count for watch timer.

[Classification]

CG_wt.c

[Syntax]

```
void WT_Stop ( void );
```

[Argument(s)]

None.

[Return value]

None.

[Example]

The example below illustrates use of the watch timer function.

[CG_main.c]

```
#include "CG_macrodriver.h"
ULONG INT_flg = 0;
void main ( void ) {
    WT_Start ();          /* Start count */
    while ( !INT_flg );
    WT_Stop ();           /* End count */
    .....
}
```

[CG_wt_user.c]

```
#include "CG_macrodriver.h"
extern ULONG INT_flg;
__interrupt void MD_INTWT ( void ) { /* Interrupt processing for INTWT */
    INT_flg = 1;
}
```

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_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 INTRTC0, then starts the cyclic interrupt function. |
| RTC_ConstPeriodInterruptDisable | Ends the cyclic interrupt function. |
| 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_IntervalStart | Starts the interval interrupt function. |
| RTC_IntervalStop | Ends the interval interrupt function. |
| RTC_IntervalInterruptEnable | Sets the cycle of the interrupts INTRTC2, then starts the interval interrupt function. |
| RTC_IntervalInterruptDisable | Ends the interval interrupt function. |
| RTC_RC1CK1HZ_OutputEnable | Enables output of the real-time counter correction clock (1 Hz) to the RC1CK1HZ pin. |
| RTC_RC1CK1HZ_OutputDisable | Disables output of the real-time counter correction clock (1 Hz) to the RC1CK1HZ pin. |
| RTC_RC1CKO_OutputEnable | Enables output of the real-time counter clock (32 kHz source) to the RC1CKO pin. |
| RTC_RC1CKO_OutputDisable | Disables output of the real-time counter clock (32 kHz source) to the RC1CKO pin. |
| RTC_RC1CKDIV_OutputEnable | Enables output of the real-time counter clock (32 kHz cycle) to the RC1CKDIV pin. |
| RTC_RC1CKDIV_OutputDisable | Disables output of the real-time counter clock (32 kHz cycle) to the RC1CKDIV pin. |
| 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_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 <i>hoursystem</i> ; | 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_macrodriver.h"
#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_macrodriver.h"
#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_macrodriver.h"
#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 INTRTC0, then starts the cyclic interrupt function.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

| I/O | Argument | Description |
|-----|-----------------------------------|--|
| I | enum RTCINTPeriod <i>period</i> ; | Interrupt INTRTC0 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 INTRTC0, then starting the cyclic interrupt function.

[CG_main.c]

```
#include "CG_macrodriver.h"
#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_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_macrodriver.h"
#include "CG_rtc.h"
void RTC_AlarmSet ( struct RTCArmValue alarmval );
```

[Argument(s)]

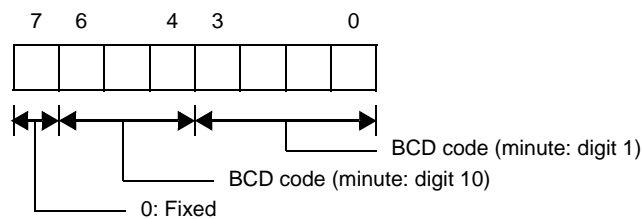
| I/O | Argument | Description |
|-----|------------------------------|--|
| I | struct RTCArmValue alarmval; | Alarm conditions (weekday, hour, minute) |

Remark Below is shown the structure RTCArmValue (alarm conditions).

```
struct RTCArmValue {
    UCHAR Alarmwm; /* Minute */
    UCHAR Alarmwh; /* Hour */
    UCHAR Alarmmw; /* Weekday */
};
```

- Alarmwm (Minute)

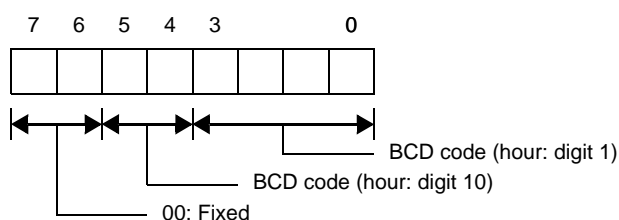
Below are shown the meanings of each bit of the structure member Alarmwm.



- Alarmwh (Hour)

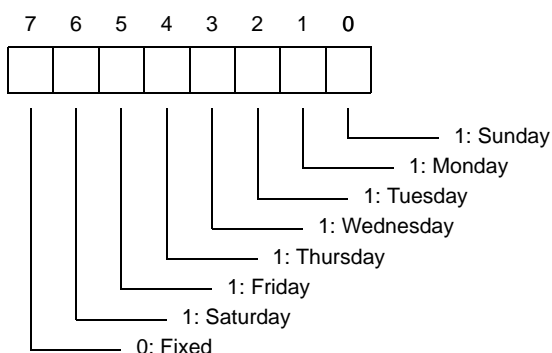
Below are shown the meanings of each bit of the structure member Alarmwh. If the real-time counter is set to the 12-hour clock, then bit 5 has the following meaning.

- 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_macrodriver.h"
#include "CG_rtc.h"
void main ( void ) {
    struct RTCArmValue 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_macrodriver.h"
#include "CG_rtc.h"
void main ( void ) {
    struct RTCArmValue alarmval;
    .....
    RTC_AlarmEnable ();          /* Start alarm interrupt function */
    .....
    alarmval.Alarmww = 0x41;
    RTC_AlarmSet ( alarmval );  /* Change conditions */
    .....
}
```

RTC_AlarmGet

Reads the alarm conditions (weekday, hour, minute).

[Classification]

CG_rtc.c

[Syntax]

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

Remark See [RTC_AlarmSet](#) for details about RTCArmValue (alarm conditions).

[Argument(s)]

| I/O | Argument | Description |
|-----|-------------------------------|--|
| O | struct RTCArmValue *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_macrodriver.h"
#include "CG_rtc.h"
void main ( void ) {
    struct RTCArmValue alarmval;
    .....
    RTC_AlarmEnable ();          /* Start alarm interrupt function */
    .....
    RTC_AlarmGet ( &alarmval ); /* Read conditions */
    .....
}
```

RTC_IntervalStart

Starts the interval interrupt function.

Remark After setting the cycle of the interrupts INTRTC2, 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 INTRTC2, then starts the interval interrupt function.

Remark Call [RTC_IntervalStart](#) to start the interval interrupt function without setting the cycle of the interrupts INTRTC2.

[Classification]

CG_rtc.c

[Syntax]

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

[Argument(s)]

| I/O | Argument | Description |
|-----|---------------------------------------|---|
| I | enum RTCINTInterval <i>interval</i> ; | Interrupt INTRTC2 cycle INTERVAL0: 2 ⁶ /fRTC INTERVAL1: 2 ⁷ /fRTC INTERVAL2: 2 ⁸ /fRTC INTERVAL3: 2 ⁹ /fRTC INTERVAL4: 2 ¹⁰ /fRTC INTERVAL5: 2 ¹¹ /fRTC INTERVAL6: 2 ¹² /fRTC |

Remark fRTC 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_macrodriver.h"
#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_RC1CK1HZ_OutputEnable

Enables output of the real-time counter correction clock (1 Hz) to the RC1CK1HZ pin.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_RC1CK1HZ_OutputEnable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_RC1CK1HZ_OutputDisable

Disables output of the real-time counter correction clock (1 Hz) to the RC1CK1HZ pin.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_RC1CK1HZ_OutputDisable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_RC1CKO_OutputEnable

Enables output of the real-time counter clock (32 kHz source) to the RC1CKO pin.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_RC1CKO_OutputEnable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_RC1CKO_OutputDisable

Disables output of the real-time counter clock (32 kHz source) to the RC1CKO pin.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_RC1CKO_OutputDisable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_RC1CKDIV_OutputEnable

Enables output of the real-time counter clock (32 kHz cycle) to the RC1CKDIV pin.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_RC1CKDIV_OutputEnable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_RC1CKDIV_OutputDisable

Disables output of the real-time counter clock (32 kHz cycle) to the RC1CKDIV pin.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_RC1CKDIV_OutputDisable ( 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_RTCCCL_OutputEnable

Enables output of the real-time counter clock (32 kHz source) to the RTCCL pin.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_RTCCCL_OutputEnable ( void );
```

[Argument(s)]

None.

[Return value]

None.

RTC_RTCCCL_OutputDisable

Disables output of the real-time counter clock (32 kHz source) to the RTCCL pin.

[Classification]

CG_rtc.c

[Syntax]

```
void    RTC_RTCCCL_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 <i>timing</i> ; | When clock errors are corrected EVERY20S: When the seconds digits are 00, 20 or 40 EVERY60S: When the seconds digits are 00 |
| I | UCHAR <i>corectval</i> ; | 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 Real-Time Output

Below is a list of API functions output by Code Generator as the real-time output function.

Table C-12. API Functions: [Real-Time Output]

| API Function Name | Function |
|----------------------|--|
| RTOn_Init | Performs initialization necessary to control real-time output functions. |
| RTOn_UserInit | Performs user-defined initialization relating to the real-time output. |
| RTOn_Enable | Enables (validates) real-time output. |
| RTOn_Disable | Disables (invalidates) real-time output. |
| RTOn_Set2BitData | Sets 2-bit data for real-time output. |
| RTOn_Set4BitData | Sets 4-bit data for real-time output. |
| RTOn_Set6BitData | Sets 6-bit data for real-time output. |
| RTOn_Set8BitData | Sets 8-bit data for real-time output. |
| RTOn_SetHigh2BitData | Sets higher 2-bit data for real-time output. |
| RTOn_SetLow2BitData | Sets lower 2-bit data for real-time output. |
| RTOn_SetHigh4BitData | Sets higher 4-bit data for real-time output. |
| RTOn_SetLow4BitData | Sets lower 4-bit data for real-time output. |
| RTOn_GetValue | Reads data from real-time output. |

RTO n _Init

Performs initialization necessary to control real-time output functions.

[Classification]

CG_rto.c

[Syntax]

```
void RTO $n$ _Init ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

RTO_n_UserInit

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

Remark This API function is called as the [RTO_n_Init](#) callback routine.

[Classification]

CG_rto_user.c

[Syntax]

```
void RTOn_UserInit ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

RTO n _Enable

Enables (validates) real-time output.

[Classification]

CG_rto.c

[Syntax]

```
void RTO $n$ _Enable ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

RTO_n_Disable

Disables (invalidates) real-time output.

[Classification]

CG_rto.c

[Syntax]

```
void RTOn_CounterDisable ( void );
```

Remark *n* is the channel number.

[Argument(s)]

None.

[Return value]

None.

RTO_n_Set2BitData

Sets 2-bit data for real-time output.

[Classification]

CG_rto.c

[Syntax]

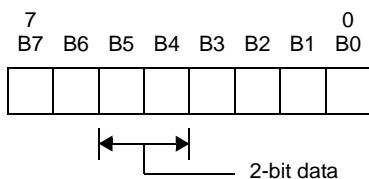
```
#include "CG_macrodriver.h"
void RTOn_Set2BitsData ( UCHAR data );
```

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|---------------------|-------------|
| I | UCHAR <i>data</i> ; | 2-bit data |

Remark The API functions treat values set in bits 4 to 5 as 2-bit data.



[Return value]

None.

RTO_n_Set4BitData

Sets 4-bit data for real-time output.

[Classification]

CG_rto.c

[Syntax]

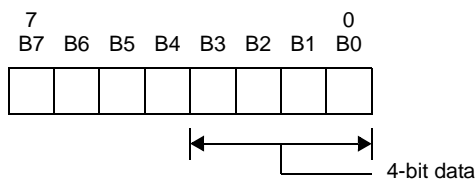
```
#include "CG_macrodriver.h"
void RTOn_Set4BitsData ( UCHAR data );
```

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|---------------------|-------------|
| I | UCHAR <i>data</i> ; | 4-bit data |

Remark The API functions treat values set in bits 0 to 3 as 4-bit data.



[Return value]

None.

RTOn_Set6BitData

Sets 6-bit data for real-time output.

[Classification]

CG_rto.c

[Syntax]

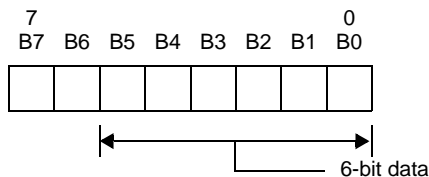
```
#include "CG_macrodriver.h"
void RTOn_Set6BitsData ( UCHAR data );
```

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|---------------------|-------------|
| I | UCHAR <i>data</i> ; | 6-bit data |

Remark The API functions treat values set in bits 0 to 5 as 6-bit data.



[Return value]

None.

RTO_n_Set8BitData

Sets 8-bit data for real-time output.

[Classification]

CG_rto.c

[Syntax]

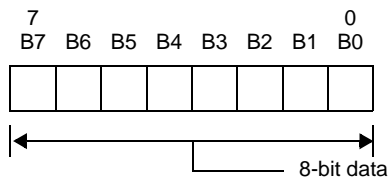
```
#include "CG_macrodriver.h"
void RTOn_Set8BitsData ( UCHAR data );
```

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|---------------------|-------------|
| I | UCHAR <i>data</i> ; | 8-bit data |

Remark The API functions treat values set in bits 0 to 7 as 8-bit data.



[Return value]

None.

RTO_n_SetHigh2BitData

Sets higher 2-bit data for real-time output.

[Classification]

CG_rto.c

[Syntax]

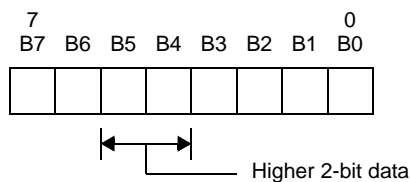
```
#include "CG_macrodriver.h"
void RTOn_SetHigh2BitsData ( UCHAR data );
```

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|---------------------|-------------------|
| I | UCHAR <i>data</i> ; | Higher 2-bit data |

Remark The API functions treat values set in bits 4 to 5 as higher 2-bit data.



[Return value]

None.

RTO_n_SetLow2BitData

Sets lower 2-bit data for real-time output.

[Classification]

CG_rto.c

[Syntax]

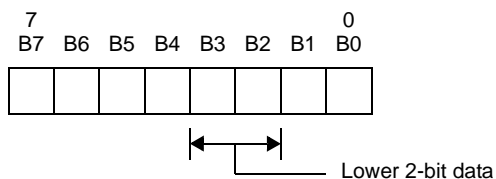
```
#include "CG_macrodriver.h"
void RTOn_SetLow2BitsData ( UCHAR data );
```

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|---------------------|------------------|
| I | UCHAR <i>data</i> ; | Lower 2-bit data |

Remark The API functions treat values set in bits 2 to 3 as lower 2-bit data.



[Return value]

None.

RTO_n_SetHigh4BitData

Sets higher 4-bit data for real-time output.

[Classification]

CG_rto.c

[Syntax]

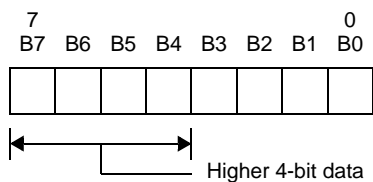
```
#include "CG_macrodriver.h"
void RTOn_SetHigh4BitsData ( UCHAR data );
```

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|---------------------|-------------------|
| I | UCHAR <i>data</i> ; | Higher 4-bit data |

Remark The API functions treat values set in bits 4 to 7 as higher 4-bit data.



[Return value]

None.

RTO_n_SetLow4BitData

Sets lower 4-bit data for real-time output.

[Classification]

CG_rto.c

[Syntax]

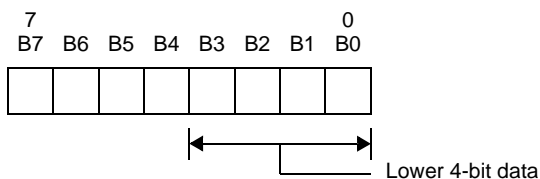
```
#include "CG_macrodriver.h"
void RTOn_SetLow4BitsData ( UCHAR data );
```

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|---------------------|------------------|
| I | UCHAR <i>data</i> ; | Lower 4-bit data |

Remark The API functions treat values set in bits 0 to 3 as lower 4-bit data.



[Return value]

None.

RTO n _GetValue

Reads data from real-time output.

[Classification]

CG_rto.c

[Syntax]

```
#include "CG_macrodriver.h"
void RTO $n$ _GetValue ( UCHAR *value );
```

Remark n is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|---------------|---|
| O | UCHAR *value; | Pointer to area in which to store the value that was read |

[Return value]

None.

[Example]

Below is an example of reading the counter value of the real-time counter.

[CG_main.c]

```
#include "CG_macrodriver.h"
void main ( void ) {
    RTO0_Set2BitData ( 0x30 );          /* Set output data */
    RTO0_Enable ();                    /* Enable real-time output */
    .....
    RTO0_Disable ();                  /* Disable real-time output */
    .....
}
```

[CG_timer_user.c]

```
#include "CG_macrodriver.h"
__interrupt void MD_INTTP4CC0 ( void ) { /* Interrupt processing for INTTP4CC0 interrupt */
    UCHAR value = 0;
    RTO0_GetValue ( &value );          /* Read output data */
    value = ~value;
    RTO0_Set2BitData ( value );        /* Set output data */
}
```

C.3.12 DMA

Below is a list of API functions output by Code Generator for DMA (Direct Memory Access) controller use.

Table C-13. API Functions: [DMA]

| API Function Name | Function |
|---|---|
| DMAAn_Init | Performs initialization necessary to control DMA controller functions. |
| DMAAn_UserInit | Performs user-defined initialization relating to the DMA controller. |
| DMAAn_Enable | Enables operation of channel <i>n</i> . |
| DMAAn_Disable | Disables operation of channel <i>n</i> . |
| DMAAn_CheckStatus | Reads the transfer status (transfer complete/transfer ongoing). |
| DMAAn_SetData | Sets the RAM address of the transfer source/destination, and the number of times the data has been transferred. |
| DMAAn_SoftwareTriggerOn | Uses a software trigger as a DMA transfer start trigger. |

DMA n _Init

Performs initialization necessary to control DMA controller functions.

[Classification]

CG_dma.c

[Syntax]

```
void DMA $n$ _Init ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

DMA n _UserInit

Performs user-defined initialization relating to the DMA controller.

Remark This API function is called as the [DMA \$n\$ _Init](#) callback routine.

[Classification]

CG_dma_user.c

[Syntax]

```
void DMA $n$ _UserInit ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

DMA n _Enable

Enables operation of channel n .

[Classification]

CG_dma.c

[Syntax]

```
void DMA $n$ _Enable ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

DMA n _Disable

Disables operation of channel n .

- Remarks 1.** This API function does not forcibly terminate DMA transfer.
- 2.** Before using this API function, you must confirm that transmission has ended via [DMA \$n\$ _CheckStatus](#).

[Classification]

CG_dma.c

[Syntax]

```
void DMA $n$ _Disable ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

[Example]

The example below shows setting the operation mode of channel 0 to "disabled".

[CG_main.c]

```
#include "CG_macrodriver.h"
void main ( void ) {
    .....
    while ( MD_COMPLETED == DMA0_CheckStatus () ); /* Check transfer status */
    DMA0_Disable (); /* Change to operation disabled status */
    .....
}
```

DMA n _CheckStatus

Reads the transfer status (transfer complete/transfer ongoing).

[Classification]

CG_dma.c

[Syntax]

```
#include "CG_macrodriver.h"
MD_STATUS DMA $n$ _CheckStatus ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

| Macro | Description |
|--------------|-------------------|
| MD_UNDEREXEC | Transfer ongoing |
| MD_COMPLETED | Transfer complete |

DMAn_SetData

Sets the RAM address of the transfer source/destination, and the number of times the data has been transferred.

Remark Calling this API function while a transfer is ongoing will end the transfer.

[Classification]

CG_dma.c

[Syntax]

```
#include "CG_macrodriver.h"
MD_STATUS DMAn_SetData ( UINT srcaddr, UINT dstaddr, UINT count );
```

Remark *n* is the channel number.

[Argument(s)]

| I/O | Argument | Description |
|-----|-----------------------|--|
| I | UINT <i>srcaddr</i> ; | RAM address of source |
| I | UINT <i>dstaddr</i> ; | RAM address of destination |
| I | UINT <i>count</i> ; | Number of data transmissions (1 to 1024) |

[Return value]

| Macro | Description |
|-------------|--------------------------------|
| MD_OK | Normal completion |
| MD_ARGERROR | Invalid argument specification |

DMA n _SoftwareTriggerOn

Uses a software trigger as a DMA transfer start trigger.

Remark After this API function is called, DMA transfer will begin if the start DMA transfer software trigger flag STG n is set to "1," or the interrupt (e.g. INTP n or INTAD) occurs.

[Classification]

CG_dma.c

[Syntax]

```
void DMA $n$ _SoftwareTriggerOn ( void );
```

Remark n is the channel number.

[Argument(s)]

None.

[Return value]

None.

[Example]

Below is an example of software trigger as a DMA transfer start trigger.

[CG_main.c]

```
void main ( void ) {  
    .....  
    DMA0_Enable ();          /* Change to operation enabled status */  
    DMA0_SoftwareTriggerOn (); /* Start DMA transfer */  
    .....  
}
```

C.3.13 LVI

Below is a list of API functions output by Code Generator for low-voltage detector use.

Table C-14. 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_Start | Starts low-voltage detection. |
| LVI_Stop | Stops low-voltage detection. |

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]

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

[Argument(s)]

None.

[Return value]

| Macro | Description |
|----------|--|
| MD_OK | Normal completion |
| MD_ERROR | Exit with error (abend) <ul style="list-style-type: none"> - The program is configured to not use the low-voltage detector function. - The object of low voltage detection is external voltage (VDD), and power voltage (VDD) <= detected voltage (VLVI). - The object of low voltage detection is external input voltage (EXLVI), and external input voltage (EXLVI) <= detected voltage (VEXLVI). |

LVI_Start

Starts low-voltage detection.

[Classification]

CG_lvi.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

| Macro | Description |
|-------------|--|
| MD_OK | Normal completion |
| MD_ARGERROR | Exit with error (abend) <ul style="list-style-type: none"> - The program is configured to not use the low-voltage detector function. - The object of low voltage detection is external voltage (VDD), and power voltage (VDD) <= detected voltage (VLVI). - The object of low voltage detection is external input voltage (EXLVI), and external input voltage (EXLVI) <= detected voltage (VEXLVI). |

LVI_Stop

Stops low-voltage detection.

[Classification]

CG_lvi.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.

APPENDIX D INDEX

A

A/D Converter ... 216
 AD_Init ... 217
 AD_Read ... 223
 AD_ReadByte ... 224
 AD_SelectADChannel ... 221
 AD_SetPFTCondition ... 222
 AD_Start ... 219
 AD_Stop ... 220
 AD_UserInit ... 218
 AD_Init ... 217
 AD_Read ... 223
 AD_ReadByte ... 224
 AD_SelectADChannel ... 221
 AD_SetPFTCondition ... 222
 AD_Start ... 219
 AD_Stop ... 220
 AD_UserInit ... 218
 API functions ... 91
 A/D Converter ... 216
 D/A Converter ... 225
 DMA ... 347
 External Bus ... 116
 Interrupt ... 125
 LVI ... 355
 Port ... 119
 Real-time Clock ... 296
 Real-Time Output ... 333
 Serial ... 136
 System ... 102
 Timer ... 231
 Watch Timer ... 291

B

Browse For Folder dialog box ... 81
 BUS_Init ... 117
 BUS_UserInit ... 118

C

CG_ChangeClockMode ... 106
 CG_ChangeFrequency ... 107
 CG_ReadResetSource ... 105
 CG_SelectPIIMode ... 110
 CG_SelectPowerSaveMode ... 108
 CG_SelectSSCGMode ... 111
 CG_SelectStabTime ... 109
 CLOCK_Init ... 103
 CLOCK_UserInit ... 104
 Code Generator panel ... 69
 Code Generator Preview panel ... 72
 Column Chooser dialog box ... 77
 CRC_GetResult ... 115
 CRC_SetData ... 114
 CRC_Start ... 113
 CSIBn_ErrorCallback ... 180
 CSIBn_Init ... 171
 CSIBn_ReceiveData ... 176
 CSIBn_ReceiveEndCallback ... 179
 CSIBn_SendData ... 175
 CSIBn_SendEndCallback ... 178
 CSIBn_SendReceiveData ... 177
 CSIBn_Start ... 173
 CSIBn_Stop ... 174
 CSIBn_UserInit ... 172
 CSIEn_ErrorCallback ... 190
 CSIEn_Init ... 181
 CSIEn_ReceiveData ... 186
 CSIEn_ReceiveEndCallback ... 189
 CSIEn_SendData ... 185
 CSIEn_SendEndCallback ... 188
 CSIEn_SendReceiveData ... 187
 CSIEn_Start ... 183
 CSIEn_Stop ... 184
 CSIEn_UserInit ... 182
 CSIFn_ErrorCallback ... 200
 CSIFn_Init ... 191

CSIFn_ReceiveData ... 196
 CSIFn_ReceiveEndCallback ... 199
 CSIFn_SendData ... 195
 CSIFn_SendEndCallback ... 198
 CSIFn_SendReceiveData ... 197
 CSIFn_Start ... 193
 CSIFn_Stop ... 194
 CSIFn_UserInit ... 192

D

D/A Converter ... 225
 DAn_Init ... 226
 DAn_SetValue ... 230
 DAn_Start ... 228
 DAn_Stop ... 229
 DAn_UserInit ... 227

DAn_Init ... 226
 DAn_SetValue ... 230
 DAn_Start ... 228
 DAn_Stop ... 229
 DAn_UserInit ... 227

[Device Pin List Information] tab ... 49

Device Pin List panel ... 58
 [External Peripheral] tab ... 64
 [Macro] tab ... 62
 [Pin Number] tab ... 60

Device Top View panel ... 66

[Device Top View Settings] tab ... 50

DMA ... 347
 DMAAn_CheckStatus ... 352
 DMAAn_Disable ... 351
 DMAAn_Enable ... 350
 DMAAn_Init ... 348
 DMAAn_SetData ... 353
 DMAAn_SoftwareTriggerOn ... 354
 DMAAn_UserInit ... 349

DMAAn_CheckStatus ... 352
 DMAAn_Disable ... 351
 DMAAn_Enable ... 350
 DMAAn_Init ... 348
 DMAAn_SetData ... 353

DMAAn_SoftwareTriggerOn ... 354
 DMAAn_UserInit ... 349

E

External Bus ... 116
 BUS_Init ... 117
 BUS_UserInit ... 118

[External Peripheral] tab ... 64

F

[File Setting] tab ... 57

Functions ... 8, 22
 Code Generator ... 22
 Pin Configurator ... 8

G

[Generation] tab ... 53

I

IIC0n_GetStopConditionCallback ... 215
 IIC0n_Init ... 201
 IIC0n_MasterErrorCallback ... 209
 IIC0n_MasterReceiveEndCallback ... 208
 IIC0n_MasterReceiveStart ... 206
 IIC0n_MasterSendEndCallback ... 207
 IIC0n_MasterSendStart ... 205
 IIC0n_SlaveErrorCallback ... 214
 IIC0n_SlaveReceiveEndCallback ... 213
 IIC0n_SlaveReceiveStart ... 211
 IIC0n_SlaveSendEndCallback ... 212
 IIC0n_SlaveSendStart ... 210
 IIC0n_Stop ... 203
 IIC0n_StopCondition ... 204
 IIC0n_UserInit ... 202

Interrupt ... 125
 INT_MaskableInterruptEnable ... 130
 INTP_Init ... 126
 INTPn_Disable ... 132
 INTPn_Enable ... 133
 INTP_UserInit ... 127
 KEY_Disable ... 134
 KEY_Enable ... 135

- KEY_Init ... 128
- KEY_UserInit ... 129
- INT_MaskableInterruptEnable ... 130
- INTP_Init ... 126
- INTPn_Disable ... 132
- INTPn_Enable ... 133
- INTP_UserInit ... 127

- K**
- KEY_Disable ... 134
- KEY_Enable ... 135
- KEY_Init ... 128
- KEY_UserInit ... 129

- L**
- LVI ... 355
 - LVI_Init ... 356
 - LVI_InterruptModeStart ... 358
 - LVI_ResetModeStart ... 359
 - LVI_Start ... 360
 - LVI_Stop ... 361
 - LVI_UserInit ... 357
- LVI_Init ... 356
- LVI_InterruptModeStart ... 358
- LVI_ResetModeStart ... 359
- LVI_Start ... 360
- LVI_Stop ... 361
- LVI_UserInit ... 357

- M**
- [Macro Setting] tab ... 56
- [Macro] tab ... 62
- Main window ... 39

- N**
- New Column dialog box ... 80

- O**
- Output panel ... 75

- P**
- [Pin Configurator Settings] tab ... 48
- [Pin Number] tab ... 60
- Port ... 119
 - PORT_ChangePmnInput ... 122
 - PORT_ChangePmnOutput ... 123
 - PORT_Init ... 120
 - PORT_UserInit ... 121
- PORT_ChangePmnInput ... 122
- PORT_ChangePmnOutput ... 123
- PORT_Init ... 120
- PORT_UserInit ... 121
- Project Tree panel ... 42
- Property panel ... 45
 - [Device Pin List Information] tab ... 49
 - [Device Top View Settings] tab ... 50
 - [File Setting] tab ... 57
 - [Generation] tab ... 53
 - [Macro Setting] tab ... 56
 - [Pin Configurator Settings] tab ... 48

- R**
- Real-time Clock ... 296
 - RTC_AlarmDisable ... 310
 - RTC_AlarmEnable ... 309
 - RTC_AlarmGet ... 314
 - RTC_AlarmSet ... 311
 - RTC_ChangeCorrectionValue ... 332
 - RTC_ConstPeriodInterruptDisable ... 308
 - RTC_ConstPeriodInterruptEnable ... 307
 - RTC_CounterDisable ... 300
 - RTC_CounterEnable ... 299
 - RTC_CounterGet ... 305
 - RTC_CounterSet ... 303
 - RTC_Init ... 297
 - RTC_IntervallInterruptDisable ... 319
 - RTC_IntervallInterruptEnable ... 317
 - RTC_IntervalStart ... 315
 - RTC_IntervalStop ... 316
 - RTC_RC1CK1HZ_OutputDisable ... 321
 - RTC_RC1CK1HZ_OutputEnable ... 320
 - RTC_RC1CKDIV_OutputDisable ... 325
 - RTC_RC1CKDIV_OutputEnable ... 324

- RTC_RC1CKO_OutputDisable ... 323
 - RTC_RC1CKO_OutputEnable ... 322
 - RTC_RTC1HZ_OutputDisable ... 327
 - RTC_RTC1HZ_OutputEnable ... 326
 - RTC_RTCCL_OutputDisable ... 329
 - RTC_RTCCL_OutputEnable ... 328
 - RTC_RTCDIV_OutputDisable ... 331
 - RTC_RTCDIV_OutputEnable ... 330
 - RTC_SetHourSystem ... 301
 - RTC_UserInit ... 298
 - Real-Time Output ... 333
 - RTOOn_Disable ... 337
 - RTOOn_Enable ... 336
 - RTOOn_GetValue ... 346
 - RTOOn_Init ... 334
 - RTOOn_Set2BitData ... 338
 - RTOOn_Set4BitData ... 339
 - RTOOn_Set6BitData ... 340
 - RTOOn_Set8BitData ... 341
 - RTOOn_SetHigh2BitData ... 342
 - RTOOn_SetHigh4BitData ... 344
 - RTOOn_SetLow2BitData ... 343
 - RTOOn_SetLow4BitData ... 345
 - RTOOn_UserInit ... 335
 - RTC_AlarmDisable ... 310
 - RTC_AlarmEnable ... 309
 - RTC_AlarmGet ... 314
 - RTC_AlarmSet ... 311
 - RTC_ChangeCorrectionValue ... 332
 - RTC_ConstPeriodInterruptDisable ... 308
 - RTC_ConstPeriodInterruptEnable ... 307
 - RTC_CounterDisable ... 300
 - RTC_CounterEnable ... 299
 - RTC_CounterGet ... 305
 - RTC_CounterSet ... 303
 - RTC_Init ... 297
 - RTC_IntervallInterruptDisable ... 319
 - RTC_IntervallInterruptEnable ... 317
 - RTC_IntervalStart ... 315
 - RTC_IntervalStop ... 316
 - RTC_RC1CK1HZ_OutputDisable ... 321
 - RTC_RC1CK1HZ_OutputEnable ... 320
 - RTC_RC1CKDIV_OutputDisable ... 325
 - RTC_RC1CKDIV_OutputEnable ... 324
 - RTC_RC1CKO_OutputDisable ... 323
 - RTC_RC1CKO_OutputEnable ... 322
 - RTC_RTC1HZ_OutputDisable ... 327
 - RTC_RTC1HZ_OutputEnable ... 326
 - RTC_RTCCL_OutputDisable ... 329
 - RTC_RTCCL_OutputEnable ... 328
 - RTC_RTCDIV_OutputDisable ... 331
 - RTC_RTCDIV_OutputEnable ... 330
 - RTC_SetHourSystem ... 301
 - RTC_UserInit ... 298
 - RTOOn_Disable ... 337
 - RTOOn_Enable ... 336
 - RTOOn_GetValue ... 346
 - RTOOn_Init ... 334
 - RTOOn_Set2BitData ... 338
 - RTOOn_Set4BitData ... 339
 - RTOOn_Set6BitData ... 340
 - RTOOn_Set8BitData ... 341
 - RTOOn_SetHigh2BitData ... 342
 - RTOOn_SetHigh4BitData ... 344
 - RTOOn_SetLow2BitData ... 343
 - RTOOn_SetLow4BitData ... 345
 - RTOOn_UserInit ... 335
- S**
- Save As dialog box ... 82
 - Serial ... 136
 - CSIBn_ErrorCallback ... 180
 - CSIBn_Init ... 171
 - CSIBn_ReceiveData ... 176
 - CSIBn_ReceiveEndCallback ... 179
 - CSIBn_SendData ... 175
 - CSIBn_SendEndCallback ... 178
 - CSIBn_SendReceiveData ... 177
 - CSIBn_Start ... 173
 - CSIBn_Stop ... 174
 - CSIBn_UserInit ... 172
 - CSIEn_ErrorCallback ... 190

| | |
|------------------------------------|-----|
| CSIFn_Init ... | 181 |
| CSIFn_ReceiveData ... | 186 |
| CSIFn_ReceiveEndCallback ... | 189 |
| CSIFn_SendData ... | 185 |
| CSIFn_SendEndCallback ... | 188 |
| CSIFn_SendReceiveData ... | 187 |
| CSIFn_Start ... | 183 |
| CSIFn_Stop ... | 184 |
| CSIFn_UserInit ... | 182 |
| CSIFn_ErrorCallback ... | 200 |
| CSIFn_Init ... | 191 |
| CSIFn_ReceiveData ... | 196 |
| CSIFn_ReceiveEndCallback ... | 199 |
| CSIFn_SendData ... | 195 |
| CSIFn_SendEndCallback ... | 198 |
| CSIFn_SendReceiveData ... | 197 |
| CSIFn_Start ... | 193 |
| CSIFn_Stop ... | 194 |
| CSIFn_UserInit ... | 192 |
| IIC0n_GetStopConditionCallback ... | 215 |
| IIC0n_Init ... | 201 |
| IIC0n_MasterErrorCallback ... | 209 |
| IIC0n_MasterReceiveEndCallback ... | 208 |
| IIC0n_MasterReceiveStart ... | 206 |
| IIC0n_MasterSendEndCallback ... | 207 |
| IIC0n_MasterSendStart ... | 205 |
| IIC0n_SlaveErrorCallback ... | 214 |
| IIC0n_SlaveReceiveEndCallback ... | 213 |
| IIC0n_SlaveReceiveStart ... | 211 |
| IIC0n_SlaveSendEndCallback ... | 212 |
| IIC0n_SlaveSendStart ... | 210 |
| IIC0n_Stop ... | 203 |
| IIC0n_StopCondition ... | 204 |
| IIC0n_UserInit ... | 202 |
| UARTAn_ErrorCallback ... | 147 |
| UARTAn_Init ... | 139 |
| UARTAn_ReceiveData ... | 144 |
| UARTAn_ReceiveEndCallback ... | 146 |
| UARTAn_SendData ... | 143 |
| UARTAn_SendEndCallback ... | 145 |
| UARTAn_SoftOverRunCallback ... | 148 |
| UARTAn_Start ... | 141 |
| UARTAn_Stop ... | 142 |
| UARTAn_UserInit ... | 140 |
| UARTBn_FIFOErrorCallback ... | 158 |
| UARTBn_Init ... | 149 |
| UARTBn_ReceiveData ... | 154 |
| UARTBn_ReceiveEndCallback ... | 156 |
| UARTBn_SendData ... | 153 |
| UARTBn_SendEndCallback ... | 155 |
| UARTBn_SingleErrorCallback ... | 157 |
| UARTBn_SoftOverRunCallback ... | 160 |
| UARTBn_Start ... | 151 |
| UARTBn_Stop ... | 152 |
| UARTBn_TimeoutErrorCallback ... | 159 |
| UARTBn_UserInit ... | 150 |
| UARTCn_ErrorCallback ... | 169 |
| UARTCn_Init ... | 161 |
| UARTCn_ReceiveData ... | 166 |
| UARTCn_ReceiveEndCallback ... | 168 |
| UARTCn_SendData ... | 165 |
| UARTCn_SendEndCallback ... | 167 |
| UARTCn_SoftOverRunCallback ... | 170 |
| UARTCn_Start ... | 163 |
| UARTCn_Stop ... | 164 |
| UARTCn_UserInit ... | 162 |
| System ... | 102 |
| CG_ChangeClockMode ... | 106 |
| CG_ChangeFrequency ... | 107 |
| CG_ReadResetSource ... | 105 |
| CG_SelectPIIMode ... | 110 |
| CG_SelectPowerSaveMode ... | 108 |
| CG_SelectSSCGMode ... | 111 |
| CG_SelectStabTime ... | 109 |
| CLOCK_Init ... | 103 |
| CLOCK_UserInit ... | 104 |
| CRC_GetResult ... | 115 |
| CRC_SetData ... | 114 |
| CRC_Start ... | 113 |
| WDT2_Restart ... | 112 |

T

- TAAAn_ChangeDuty ... 262
- TAAAn_ChangeTimerCondition ... 258
- TAAAn_ControlOutputToggle ... 259
- TAAAn_GetFreeRunningValue ... 261
- TAAAn_GetPulseWidth ... 260
- TAAAn_Init ... 254
- TAAAn_SoftwareTriggerOn ... 263
- TAAAn_Start ... 256
- TAAAn_Stop ... 257
- TAAAn_UserInit ... 255
- TABn_ChangeDuty ... 272
- TABn_ChangeTimerCondition ... 268
- TABn_ControlOutputToggle ... 269
- TABn_GetFreeRunningValue ... 271
- TABn_GetPulseWidth ... 270
- TABn_Init ... 264
- TABn_SoftwareTriggerOn ... 273
- TABn_Start ... 266
- TABn_Stop ... 267
- TABn_UserInit ... 265
- Timer ... 231
 - TAAAn_ChangeDuty ... 262
 - TAAAn_ChangeTimerCondition ... 258
 - TAAAn_ControlOutputToggle ... 259
 - TAAAn_GetFreeRunningValue ... 261
 - TAAAn_GetPulseWidth ... 260
 - TAAAn_Init ... 254
 - TAAAn_SoftwareTriggerOn ... 263
 - TAAAn_Start ... 256
 - TAAAn_Stop ... 257
 - TAAAn_UserInit ... 255
 - TABn_ChangeDuty ... 272
 - TABn_ChangeTimerCondition ... 268
 - TABn_ControlOutputToggle ... 269
 - TABn_GetFreeRunningValue ... 271
 - TABn_GetPulseWidth ... 270
 - TABn_Init ... 264
 - TABn_SoftwareTriggerOn ... 273
 - TABn_Start ... 266
 - TABn_Stop ... 267
 - TABn_UserInit ... 265
- TABn_UserInit ... 265
- TAMPn_ChangeDuty ... 241
- TAMPn_ChangeTimerCondition ... 237
- TAMPn_GetFreeRunningValue ... 240
- TAMPn_GetPulseWidth ... 239
- TAMPn_Init ... 233
- TAMPn_SoftwareTriggerOn ... 242
- TAMPn_Start ... 235
- TAMPn_Stop ... 236
- TAMPn_UserInit ... 234
- TMQ0_ChangeDuty ... 251
- TMQ0_ChangeTimerCondition ... 247
- TMQ0_GetFreeRunningValue ... 250
- TMQ0_GetPulseWidth ... 249
- TMQ0_Init ... 243
- TMQ0_SoftwareTriggerOn ... 253
- TMQ0_Start ... 245
- TMQ0_Stop ... 246
- TMQ0_UserInit ... 244
- TMT0_ChangeCountValue ... 285
- TMT0_ChangeDuty ... 281
- TMT0_ChangeTimerCondition ... 278
- TMT0_DisableHold ... 284
- TMT0_EnableHold ... 283
- TMT0_GetFreeRunningValue ... 280
- TMT0_GetPulseWidth ... 279
- TMT0_Init ... 274
- TMT0_SoftwareTriggerOn ... 282
- TMT0_Start ... 276
- TMT0_Stop ... 277
- TMT0_UserInit ... 275
- TMMn_ChangeTimerCondition ... 290
- TMMn_Init ... 286
- TMMn_Start ... 288
- TMMn_Stop ... 289
- TMMn_UserInit ... 287

- TMPn_ChangeDuty ... 241
 TMPn_ChangeTimerCondition ... 237
 TMPn_GetFreeRunningValue ... 240
 TMPn_GetPulseWidth ... 239
 TMPn_Init ... 233
 TMPn_SoftwareTriggerOn ... 242
 TMPn_Start ... 235
 TMPn_Stop ... 236
 TMPn_UserInit ... 234
 TMQ0_ChangeDuty ... 251
 TMQ0_ChangeTimerCondition ... 247
 TMQ0_GetFreeRunningValue ... 250
 TMQ0_GetPulseWidth ... 249
 TMQ0_Init ... 243
 TMQ0_SoftwareTriggerOn ... 253
 TMQ0_Start ... 245
 TMQ0_Stop ... 246
 TMQ0_UserInit ... 244
 TMT0_ChangeCountValue ... 285
 TMT0_ChangeDuty ... 281
 TMT0_ChangeTimerCondition ... 278
 TMT0_DisableHold ... 284
 TMT0_EnableHold ... 283
 TMT0_GetFreeRunningValue ... 280
 TMT0_GetPulseWidth ... 279
 TMT0_Init ... 274
 TMT0_SoftwareTriggerOn ... 282
 TMT0_Start ... 276
 TMT0_Stop ... 277
 TMT0_UserInit ... 275
- U**
- UARTAn_ErrorCallback ... 147
 UARTAn_Init ... 139
 UARTAn_ReceiveData ... 144
 UARTAn_ReceiveEndCallback ... 146
 UARTAn_SendData ... 143
 UARTAn_SendEndCallback ... 145
 UARTAn_SoftOverRunCallback ... 148
 UARTAn_Start ... 141
 UARTAn_Stop ... 142
- UARTAn_UserInit ... 140
 UARTBn_FIFOErrorCallback ... 158
 UARTBn_Init ... 149
 UARTBn_ReceiveData ... 154
 UARTBn_ReceiveEndCallback ... 156
 UARTBn_SendData ... 153
 UARTBn_SendEndCallback ... 155
 UARTBn_SingleErrorCallback ... 157
 UARTBn_SoftOverRunCallback ... 160
 UARTBn_Start ... 151
 UARTBn_Stop ... 152
 UARTBn_TimeoutErrorCallback ... 159
 UARTBn_UserInit ... 150
 UARTCn_ErrorCallback ... 169
 UARTCn_Init ... 161
 UARTCn_ReceiveData ... 166
 UARTCn_ReceiveEndCallback ... 168
 UARTCn_SendData ... 165
 UARTCn_SendEndCallback ... 167
 UARTCn_SoftOverRunCallback ... 170
 UARTCn_Start ... 163
 UARTCn_Stop ... 164
 UARTCn_UserInit ... 162
- W**
- Watch Timer ... 291
 - WT_Init ... 292
 - WT_Start ... 294
 - WT_Stop ... 295
 - WT_UserInit ... 293
- WDT2_Restart ... 112
 Window reference ... 38
 WT_Init ... 292
 WT_Start ... 294
 WT_Stop ... 295
 WT_UserInit ... 293

Revision Record

| Rev. | Date | Description | |
|------|--------------|-------------|----------------------|
| | | Page | Summary |
| 1.00 | Sep 01, 2012 | - | First Edition issued |

CubeSuite+ V1.03.00 User's Manual:
V850 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

CubeSuite+ V1.03.00