CubeSuite+ V1.03.00
Integrated Development Environment
User's Manual: V850 Design

Target Device
V850 Family
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 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 implants 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: Higher 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 ... XXX
Hexadecimal ... 0xXXXX
### Related Documents

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

<table>
<thead>
<tr>
<th>Document Name</th>
<th>Document No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>R20UT2135E</td>
</tr>
<tr>
<td>V850 Design</td>
<td>R20UT2136E</td>
</tr>
<tr>
<td>R8C Design</td>
<td>R20UT2137E</td>
</tr>
<tr>
<td>RL78 Design</td>
<td>R20UT2138E</td>
</tr>
<tr>
<td>78K0R Design</td>
<td>R20UT0767E</td>
</tr>
<tr>
<td>78K0 Design</td>
<td>R20UT0768E</td>
</tr>
<tr>
<td>RX Coding</td>
<td>R20UT0783E</td>
</tr>
<tr>
<td>V850 Coding</td>
<td>R20UT0732E</td>
</tr>
<tr>
<td>Coding for CX Compiler</td>
<td>R20UT2139E</td>
</tr>
<tr>
<td>R8C Coding</td>
<td>R20UT0576E</td>
</tr>
<tr>
<td>RL78, 78K0R Coding</td>
<td>R20UT2140E</td>
</tr>
<tr>
<td>78K0 Coding</td>
<td>R20UT2141E</td>
</tr>
<tr>
<td>RX Build</td>
<td>R20UT0783E</td>
</tr>
<tr>
<td>V850 Build</td>
<td>R20UT0732E</td>
</tr>
<tr>
<td>Build for CX Compiler</td>
<td>R20UT2142E</td>
</tr>
<tr>
<td>R8C Build</td>
<td>R20UT0783E</td>
</tr>
<tr>
<td>RL78, 78K0R Build</td>
<td>R20UT2143E</td>
</tr>
<tr>
<td>78K0 Build</td>
<td>R20UT0783E</td>
</tr>
<tr>
<td>RX Debug</td>
<td>R20UT2145E</td>
</tr>
<tr>
<td>V850 Debug</td>
<td>R20UT2146E</td>
</tr>
<tr>
<td>R8C Debug</td>
<td>R20UT2147E</td>
</tr>
<tr>
<td>RL78 Debug</td>
<td>R20UT2148E</td>
</tr>
<tr>
<td>78K0R Debug</td>
<td>R20UT2149E</td>
</tr>
<tr>
<td>78K0 Debug</td>
<td>R20UT2150E</td>
</tr>
<tr>
<td>Analysis</td>
<td>R20UT2151E</td>
</tr>
<tr>
<td>Message</td>
<td>R20UT2152E</td>
</tr>
</tbody>
</table>

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

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

CHAPTER 1  GENERAL  ...  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
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.


(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.


(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](image)

**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.

![Column Chooser dialog box](Figure 2-2. Select Item)

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

<table>
<thead>
<tr>
<th>Table 2-1. Select Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checked</td>
</tr>
<tr>
<td>Not checked</td>
</tr>
</tbody>
</table>
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**

![Change Display Order](image)

**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.

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.

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.

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](image)

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.

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

**Table 2-2. Select Color**

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

<table>
<thead>
<tr>
<th>Popup Information</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display all</td>
<td>Displays the &quot;Description&quot;, &quot;Recommend Connection for Unused&quot;, and &quot;Attention&quot; strings for the device pin list.</td>
</tr>
<tr>
<td>Description / recommended connection for unused pin only</td>
<td>Displays the &quot;Description&quot;, and &quot;Recommend Connection for Unused&quot; strings for the device pin list.</td>
</tr>
<tr>
<td>Attention only</td>
<td>Displays the &quot;Attention&quot; string for the device pin list.</td>
</tr>
<tr>
<td>Not display</td>
<td>Hides tooltips when the mouse cursor hovers over a pin.</td>
</tr>
</tbody>
</table>
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**

![Figure 2-10](image)

**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).

<table>
<thead>
<tr>
<th>Display</th>
<th>Displays the &quot;Define Name&quot; string of the device pin list in appended format.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not display</td>
<td>Hides the &quot;Define Name&quot; string of the device pin list.</td>
</tr>
</tbody>
</table>

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).

<table>
<thead>
<tr>
<th>Display all</th>
<th>Displays functions selected via the device pin list's &quot;Function&quot; feature in parentheses.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selected function only</td>
<td>Only display functions selected via the device pin list's &quot;Function&quot; feature in the device top view.</td>
</tr>
</tbody>
</table>
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 ...].

![Save As dialog box]

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


(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.


(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.

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>
<thead>
<tr>
<th>Character Set</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII</td>
<td>1-byte alphabet, number, symbol</td>
</tr>
<tr>
<td>Shift-JIS</td>
<td>2-byte alphabet, number, symbol, Hiragana, Katakana, Kanji and 1-byte Katakana</td>
</tr>
<tr>
<td>EUC-JP</td>
<td>2-byte alphabet, number, symbol, Hiragana, Katakana, Kanji and 1-byte Katakana</td>
</tr>
<tr>
<td>UTF-8</td>
<td>2-byte alphabet, number, symbol, Hiragana, Katakana, Kanji (include Chinese character) and 1-byte Katakana</td>
</tr>
</tbody>
</table>

(2) Number

Notations allowed when entering numbers are as follows.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal number</td>
<td>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</td>
</tr>
<tr>
<td>Hex number</td>
<td>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)</td>
</tr>
</tbody>
</table>
3.3.2 Icon indicating incorrect entry

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

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

Figure 3-2. Icon Indicating Incorrect Entry
3.3.3 Icon indicating pin conflict

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

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

Figure 3-3. Icon Indicating Pin Conflict
3.4 Confirm Source Code

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

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

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>
<thead>
<tr>
<th>Color</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Green</td>
<td>Comment</td>
</tr>
<tr>
<td>Blue</td>
<td>Reserved word for C compiler</td>
</tr>
<tr>
<td>Red</td>
<td>Numeric value</td>
</tr>
<tr>
<td>Black</td>
<td>Code section</td>
</tr>
<tr>
<td>Gray</td>
<td>File name</td>
</tr>
</tbody>
</table>

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

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

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

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

Figure 3-5. Output Source Code

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

Figure 3-6. Configure Whether to Register
3.5.1 Setting that determines whether or not to generate source code

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

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

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

Figure 3-8. Setting That Determines Whether or Not to Generate Source Code
Remark  You can confirm the current setting for the generation of source code by checking the type of icon in the Code Generator Preview panel.

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

<table>
<thead>
<tr>
<th>Type of Icon</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="icon1" alt="Icon" /></td>
<td>Source code for the currently selected API function is generated.</td>
</tr>
<tr>
<td><img src="icon2" alt="Icon" /></td>
<td>Source code for the currently selected API function is generated.</td>
</tr>
<tr>
<td><img src="icon3" alt="Icon" /></td>
<td>Source code for the currently selected API function is not generated.</td>
</tr>
</tbody>
</table>

#### 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**

![Code Generator Preview](code_generator_preview.png)

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>
<thead>
<tr>
<th>Output Mode</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do nothing if file exists</td>
<td>If a file with the same name exists, a new file will not be output.</td>
</tr>
<tr>
<td>Merge file</td>
<td>If a file with the same name exists, a new file is merged with the existing file.</td>
</tr>
<tr>
<td></td>
<td>Only the section between &quot;/* Start user code ... End user code. Do not edit comment generated here <em>/&quot; and &quot;/</em> End user code. Do not edit comment generated here */&quot; will be merged.</td>
</tr>
<tr>
<td>Overwrite file</td>
<td>If a file with the same name exists, the existing file is overwitten by a new file.</td>
</tr>
</tbody>
</table>
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].


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

<table>
<thead>
<tr>
<th>File Name</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>macro</td>
<td>A file that contains the information configured using the Code Generator</td>
</tr>
<tr>
<td>function</td>
<td>A file that contains the information regarding the source code</td>
</tr>
</tbody>
</table>

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

Figure 3-13. Output Example of Report File "macro"
**Figure 3-14. Output Example of Report File “function”**

![Function List - Microsoft Internet Explorer](image)

**MCU name:** V850ES/JJ3  
**Chip name:** uPD70F3746

<table>
<thead>
<tr>
<th>Module</th>
<th>File</th>
<th>Macro</th>
<th>Function</th>
<th>Default</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common</td>
<td>CG_main.c</td>
<td>void main(void)</td>
<td>main</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>void R_MAIN_UserInit(void)</td>
<td>R_MAIN_UserInit</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CG_systeminit.c</td>
<td>void systeminit(void)</td>
<td>CG_systeminit.c</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CG_macrodriver.h</td>
<td></td>
<td>CG_macrodriver.h</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CG_userdefine.h</td>
<td></td>
<td>CG_userdefine.h</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CG_Lk.dir</td>
<td></td>
<td>CG_Lk.dir</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CG_start.s</td>
<td></td>
<td>CG_start.s</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CG_inittab.s</td>
<td></td>
<td>CG_inittab.s</td>
<td>Used</td>
<td></td>
</tr>
<tr>
<td>System</td>
<td>CG_system.c</td>
<td>void CLOCK_Init(void)</td>
<td>CG_system.c</td>
<td>Used</td>
<td></td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Report Type</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML file</td>
<td>Outputs a report file in HTML format.</td>
</tr>
<tr>
<td>CSV file</td>
<td>Outputs a report file in CSV format.</td>
</tr>
</tbody>
</table>
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

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

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

### (b) [Edit] menu

<table>
<thead>
<tr>
<th>Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undo</td>
<td>Property panel-dedicated item&lt;br&gt;Cancels the effect of an edit operation to restore the previous state.</td>
</tr>
<tr>
<td>Cut</td>
<td>Property panel-dedicated item&lt;br&gt;Sends the character string or lines selected with range selection to the clipboard and deletes them.</td>
</tr>
<tr>
<td>Copy</td>
<td>Property panel/Output panel-dedicated item&lt;br&gt;Sends the character string or lines selected with range selection to the clipboard.</td>
</tr>
<tr>
<td>Paste</td>
<td>Property panel-dedicated item&lt;br&gt;Inserts the contents of the clipboard at the caret position.</td>
</tr>
<tr>
<td>Delete</td>
<td>Property panel-dedicated item&lt;br&gt;Deletes the character string or the lines selected with the range selection.</td>
</tr>
<tr>
<td>Select All</td>
<td>Property panel/Output panel-dedicated item&lt;br&gt;Selects all the strings displayed in the item being edited or all the strings displayed in the Message area.</td>
</tr>
</tbody>
</table>
(c) [Help] menu

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search...</td>
<td>Device Pin List panel/Code Generator Preview panel/Output panel-dedicated item</td>
</tr>
<tr>
<td></td>
<td>Opens the Search and Replace dialog box for searching strings with the [Quick Search] tab selected.</td>
</tr>
<tr>
<td>Replace...</td>
<td>Output panel-dedicated item</td>
</tr>
<tr>
<td></td>
<td>Opens the Search and Replace dialog box for replacing strings with the [Whole Replace] tab selected.</td>
</tr>
</tbody>
</table>

(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
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.
(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 disokayed.

<table>
<thead>
<tr>
<th>Node</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Pin List</td>
<td>Opens the Device Pin List panel for entering information on the pins of the microcontroller.</td>
</tr>
<tr>
<td>Device Top View</td>
<td>Opens the Device Top View panel that displays the information entered in the Device Pin List panel.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>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.</td>
</tr>
<tr>
<td>External Bus</td>
<td>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.</td>
</tr>
<tr>
<td>Port</td>
<td>Opens the [Port] for configuring the information necessary to control the port functions provided by the microcontroller.</td>
</tr>
<tr>
<td>Interrupt</td>
<td>Opens the [Interrupt] for configuring the information necessary to control the interrupt functions and the key interrupt function provided by the microcontroller.</td>
</tr>
<tr>
<td>Serial</td>
<td>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.</td>
</tr>
<tr>
<td>A/D Converter</td>
<td>Opens the [A/D Converter] for configuring the information necessary to control the function of A/D converter provided by the microcontroller.</td>
</tr>
<tr>
<td>D/A Converter</td>
<td>Opens the [D/A Converter] for configuring the information necessary to control the function of D/A converter provided by the microcontroller.</td>
</tr>
<tr>
<td>Timer</td>
<td>Opens the [Timer] for configuring the information necessary to control the functions of timer array unit provided by the microcontroller.</td>
</tr>
<tr>
<td>Watch Timer</td>
<td>Opens the [Watch Timer] for configuring the information necessary to control the functions of watch timer provided by the microcontroller.</td>
</tr>
<tr>
<td>Real-time Clock</td>
<td>Opens the [Real-time Clock] for configuring the information necessary to control the functions of real-time counter provided by the microcontroller.</td>
</tr>
<tr>
<td>Real-Time Output</td>
<td>Opens the [Real-Time Output] for configuring the information necessary to control the real-time output functions provided by the microcontroller.</td>
</tr>
<tr>
<td>DMA</td>
<td>Opens the [DMA] for configuring the information necessary to control the functions of DMA (Direct Memory Access) controller provided by the microcontroller.</td>
</tr>
<tr>
<td>LVI</td>
<td>Opens the [LVI] for configuring the information necessary to control the functions of low-voltage detector provided by the microcontroller.</td>
</tr>
</tbody>
</table>

(c) **Icons**

The table below displays the meaning of the icon displayed to the left of the string representing the peripheral function node.
## [Help] menu (Project Tree panel-dedicated items)

| 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. |

### Context menu

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

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return to Reset Value</td>
<td>Restores the information for the selected peripheral function node to its default state.</td>
</tr>
<tr>
<td>Property</td>
<td>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 &quot;[System], [Port], etc.&quot;).</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>▼</th>
<th>Indicates that the items within the category are displayed as a &quot;collapsed view&quot;.</th>
</tr>
</thead>
<tbody>
<tr>
<td>▲</td>
<td>Indicates that the items within the category are displayed as an &quot;expanded view&quot;.</td>
</tr>
</tbody>
</table>

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)]

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

[Context menu]

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

(1) While the item is being edited

<table>
<thead>
<tr>
<th>Undo</th>
<th>Cancels the effect of an edit operation to restore the previous state.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut</td>
<td>Sends the character string or lines selected with range selection to the clipboard and deletes them.</td>
</tr>
</tbody>
</table>
### 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

<table>
<thead>
<tr>
<th>Property Reset to Default</th>
<th>Restores the selected item to its default state.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property Reset All to Default</td>
<td>Restores all items to their default state.</td>
</tr>
</tbody>
</table>
[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.

<table>
<thead>
<tr>
<th>Version</th>
<th>Displays the version of the Pin Configurator (Pin Configurator Plug-in).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release date</td>
<td>Displays the release date of the Pin Configurator (Pin Configurator Plug-in).</td>
</tr>
</tbody>
</table>

(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.

<table>
<thead>
<tr>
<th>Version</th>
<th>Displays the version of Pin Configurator (Pin Configurator Plug-in).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release date</td>
<td>Displays the release date of Pin Configurator (Pin Configurator Plug-in).</td>
</tr>
</tbody>
</table>
[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.

<table>
<thead>
<tr>
<th>Power pins</th>
<th>Selects the display color for power pins (pins whose use is limited to power).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Special pins</td>
<td>Selects the display color for special pins (pins with specified uses).</td>
</tr>
</tbody>
</table>
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.

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

(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.

<table>
<thead>
<tr>
<th>Tool tip</th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display all</td>
<td>Displays the &quot;Description&quot;, &quot;Recommend Connection for Unused&quot;, and &quot;Attention&quot; strings for the device pin list.</td>
</tr>
<tr>
<td>Description / recommended connection for unused pin only</td>
<td>Displays the &quot;Description&quot;, and &quot;Recommend Connection for Unused&quot; string for the device pin list.</td>
</tr>
<tr>
<td>Attention only</td>
<td>Displays the &quot;Attention&quot; string for the device pin list.</td>
</tr>
<tr>
<td>Not display</td>
<td>Hides tooltips when the mouse cursor hovers over a pin.</td>
</tr>
</tbody>
</table>

(3) [Pin Name Display] category
Select whether to display additional information about the pin in the device top view.
### Define name

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display</td>
<td>Displays the &quot;Define Name&quot; string of the device pin list in appended format.</td>
</tr>
<tr>
<td>Not display</td>
<td>Hides the &quot;Define Name&quot; string of the device pin list.</td>
</tr>
</tbody>
</table>

### Pin function

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display all</td>
<td>Displays functions selected via the device pin list's &quot;Function&quot; feature in parentheses.</td>
</tr>
<tr>
<td>Selected function only</td>
<td>Only display functions selected via the device pin list's &quot;Function&quot; feature in the device top view.</td>
</tr>
</tbody>
</table>
[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](Image)

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.

<table>
<thead>
<tr>
<th>Version</th>
<th>Displays the version of the Code Generator (Code Library).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release date</td>
<td>Displays the release date of the Code Generator (Code Library).</td>
</tr>
</tbody>
</table>
(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.

<table>
<thead>
<tr>
<th>Output control of API function</th>
<th>Views or Selects the type of output API functions (all API functions or only initialization API functions) when the [Generate Code] button is pressed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output all API functions</td>
<td>Outputs all API functions.</td>
</tr>
<tr>
<td>according to the setting</td>
<td></td>
</tr>
<tr>
<td>Output only initialization</td>
<td>Outputs only initialization API functions.</td>
</tr>
<tr>
<td>API function</td>
<td></td>
</tr>
</tbody>
</table>

Generate file

<table>
<thead>
<tr>
<th>Generate file</th>
<th>Views or select the operation mode applied when the [Generate Code] button is pressed. Operation mode applied when you select [File] menu &gt;&gt; [Save Code Generator Report] is fixed to &quot;Overwrite file&quot;.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do nothing if file exists</td>
<td>If a file with the same name exists, a new file will not be output.</td>
</tr>
<tr>
<td>Merge file</td>
<td>If a file with the same name exists, a new file is merged with the existing file. Only the section between &quot;/* Start user code ... Do not edit comment generated here <em>/&quot; and &quot;/</em> End user code. Do not edit comment generated here */&quot; will be merged.</td>
</tr>
<tr>
<td>Overwrite file</td>
<td>If a file with the same name exists, the existing file is overwritten by a new file.</td>
</tr>
</tbody>
</table>

Output folder

<table>
<thead>
<tr>
<th>Output folder</th>
<th>Views or select the destination folder for various files (source code and report files) which are output when the [Generate Code] button is pressed or when [File] menu &gt;&gt; [Save Code Generator Report] is selected.</th>
</tr>
</thead>
</table>

Report type

<table>
<thead>
<tr>
<th>Report type</th>
<th>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 &gt;&gt; [Save Code Generator Report] is selected.</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML file</td>
<td>Outputs a report file in HTML format.</td>
</tr>
<tr>
<td>CSV file</td>
<td>Outputs a report file in CSV format.</td>
</tr>
</tbody>
</table>

Register files

<table>
<thead>
<tr>
<th>Register files</th>
<th>Selects whether source code generated by pressing the [Generate Code] button should be added to the project.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output files to project</td>
<td>Adds output source code to the project. The source code will be added to the Project Tree panel, under the [File] - [Code Generator] node.</td>
</tr>
<tr>
<td>Not output files to project</td>
<td>Does not add output source code to the project.</td>
</tr>
</tbody>
</table>

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.
Remark  If "Not reflected" is selected, then the \textbf{Reflect in Pin} 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](image)

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.

<table>
<thead>
<tr>
<th>Default name</th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>The file name is a default name. Changing this area from &quot;No&quot; to &quot;Yes&quot; changes the name of the file to its default name.</td>
</tr>
<tr>
<td>No</td>
<td>The file name is not a default name.</td>
</tr>
</tbody>
</table>

File name
Displays or change the name of the file selected on the Code Generator Preview panel.
Device Pin List panel

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

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

Figure A-11.   Device Pin List Panel

The following items are explained here.
- [How to open]
- [Description of each area]
- [(File) menu (Device Pin List panel-dedicated items)]
- [(Help) menu (Device Pin List panel-dedicated items)]

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

[Description of each area]

(1) Toolbar
This area consists of the following buttons.

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Expanded View]</td>
<td>Displays the information in the Device pin list area in an expanded view.</td>
</tr>
<tr>
<td>![Folded View]</td>
<td>Displays the information in the Device pin list area in a folded view only.</td>
</tr>
<tr>
<td>![Auto Configuration]</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Remarks 1.
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 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)]

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save Pin List</td>
<td>Saves a report file (a file containing information configured using the Pin Configurator: device pin list) overwriting the existing file.</td>
</tr>
<tr>
<td>Save Pin List As...</td>
<td>Opens the <a href="#">Save As dialog box</a> for naming and saving a report file (a file containing information configured using the Pin Configurator: device pin list).</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Help for Device Pin List Panel</td>
<td>Displays the help of this panel.</td>
</tr>
</tbody>
</table>
[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.

<table>
<thead>
<tr>
<th>Column Heading</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin Number</td>
<td>Displays the pin number of the pin.</td>
</tr>
<tr>
<td>Pin Name</td>
<td>This area allows you to select &quot;which function to use&quot; when the pin has more than one functions.</td>
</tr>
<tr>
<td>Function</td>
<td>This area allows you to select &quot;which function to use&quot; when the pin has more than one functions.</td>
</tr>
<tr>
<td>I/O</td>
<td>This area allows you to select the I/O mode of the pin.</td>
</tr>
<tr>
<td>N-ch</td>
<td>This area allows you to select &quot;which output mode to apply&quot; when using the pin in the output mode.</td>
</tr>
</tbody>
</table>
### Column Heading Outline

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

### 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.

<table>
<thead>
<tr>
<th>Column Heading</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macro Name</td>
<td>Displays the name of the peripheral function.</td>
</tr>
<tr>
<td>Total</td>
<td>Displays the total number of pins assigned to the peripheral function.</td>
</tr>
<tr>
<td>Used</td>
<td>Displays the total number of pins for which the purpose has been set.</td>
</tr>
<tr>
<td>Used in Other Macro</td>
<td>Displays the total number of pins for which the purpose has been set by other peripheral functions.</td>
</tr>
</tbody>
</table>
(b) Second layer

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

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.

<table>
<thead>
<tr>
<th>Column Heading</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>External Peripheral</td>
<td>Displays the name of the external peripheral controller.</td>
</tr>
<tr>
<td></td>
<td>To change the name, select this field and then press the [F2] key.</td>
</tr>
<tr>
<td>Total</td>
<td>Displays the total number of pins allocated for connection with the microcontroller.</td>
</tr>
</tbody>
</table>
(b) Second layer

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

**Remarks**

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

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

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

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

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

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

Figure A-15. Device Top View Panel

The following items are explained here.
- [How to open]
- [Description of each area]
- [[File] menu (Device Top View panel-dedicated items)]
- [[Help] menu (Device Top View panel-dedicated items)]
- [Context menu]

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

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

(1) Toolbar
This area consists of the following buttons.

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Icon" /></td>
<td>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.</td>
</tr>
<tr>
<td><img src="image2.png" alt="Icon" /></td>
<td>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.</td>
</tr>
<tr>
<td><img src="image3.png" alt="Icon" /></td>
<td>Rotates the content in the Device top view area 90 degrees counter-clockwise.</td>
</tr>
<tr>
<td><img src="image4.png" alt="Icon" /></td>
<td>Rotates the content in the Device top view area 90 degrees clockwise.</td>
</tr>
<tr>
<td><img src="image5.png" alt="Icon" /></td>
<td>Expands or reduces the content in the Device top view area.</td>
</tr>
</tbody>
</table>

(2) [User Define] area
Drag and drop the button from this area to the Device top view area to create 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)]

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save Top View</td>
<td>Saves a report file (a file containing information configured using the Pin Configurator: device top view) overwriting the existing file.</td>
</tr>
<tr>
<td>Save Top View As...</td>
<td>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).</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open Help for Device Top View Panel</td>
<td>Displays the help of this panel.</td>
</tr>
</tbody>
</table>

[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

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use as</td>
<td>If the pin has multiple functions, select which function to use.</td>
</tr>
<tr>
<td>Connect to External Peripheral</td>
<td>Selects which external peripheral controller to connect the pin to.</td>
</tr>
</tbody>
</table>

(2) When an external peripheral controller is right clicked

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disconnect Pin</td>
<td>Disconnects from the pin.</td>
</tr>
<tr>
<td>Delete External Peripheral</td>
<td>Removes the external peripheral controller.</td>
</tr>
</tbody>
</table>
**Code Generator panel**

This panel allows you to configure the information necessary to control the peripheral functions provided by the microcontroller.

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

<table>
<thead>
<tr>
<th><strong>(1)</strong></th>
<th><strong>(2)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Code Generator panel" /></td>
<td><img src="image" alt="Code Generator panel" /></td>
</tr>
</tbody>
</table>

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 disokayed.

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Combine with PinPart" /></td>
<td>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 &quot;Not reflected&quot; is selected in the [PinPart Combination Mode] category of the [Generation] tab.</td>
</tr>
<tr>
<td><img src="image" alt="Generate Code" /></td>
<td>Outputs the source code (device driver program) to the folder specified by selecting [Generation] tab &gt;&gt; [Output folder] in the Property panel.</td>
</tr>
<tr>
<td><img src="image" alt="" /></td>
<td>Changes the view in the Information setting area to the &quot;[System]&quot; in order to configure the information required to control the clock generation function, standby function, and etc. provided by the microcontroller.</td>
</tr>
<tr>
<td><img src="image" alt="" /></td>
<td>Changes the view in the Information setting area to the &quot;[External Bus]&quot; 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.</td>
</tr>
<tr>
<td><img src="image" alt="" /></td>
<td>Changes the content displayed in the Information setting area to the &quot;[Port]&quot; for configuring the information necessary to control the port functions provided by the microcontroller.</td>
</tr>
<tr>
<td><img src="image" alt="" /></td>
<td>Changes the content displayed in the Information setting area to the &quot;[Interrupt]&quot; for configuring the information necessary to control the external interrupt functions and the key interrupt function provided by the microcontroller.</td>
</tr>
<tr>
<td><img src="image" alt="" /></td>
<td>Changes the view in the Information setting area to the &quot;[Serial]&quot; 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.</td>
</tr>
<tr>
<td><img src="image" alt="" /></td>
<td>Changes the content displayed in the Information setting area to the &quot;[A/D Converter]&quot; for configuring the information necessary to control the function of A/D converter provided by the microcontroller.</td>
</tr>
<tr>
<td><img src="image" alt="" /></td>
<td>Changes the content displayed in the Information setting area to the &quot;[D/A Converter]&quot; for configuring the information necessary to control the function of D/A converter provided by the microcontroller.</td>
</tr>
<tr>
<td><img src="image" alt="" /></td>
<td>Changes the view in the Information setting area to the &quot;[Timer]&quot; 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.</td>
</tr>
<tr>
<td><img src="image" alt="" /></td>
<td>Changes the content displayed in the Information setting area to the &quot;[Watch Timer]&quot; for configuring the information necessary to control the functions of watchdog timer provided by the microcontroller.</td>
</tr>
<tr>
<td><img src="image" alt="" /></td>
<td>Changes the content displayed in the Information setting area to the &quot;[Real-time Clock]&quot; for configuring the information necessary to control the functions of real-time counter provided by the microcontroller.</td>
</tr>
<tr>
<td><img src="image" alt="" /></td>
<td>Changes the content displayed in the Information setting area to the &quot;[Real-Time Output]&quot; for configuring the information necessary to control the functions of real-time counter provided by the microcontroller.</td>
</tr>
</tbody>
</table>
(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.


[[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

<table>
<thead>
<tr>
<th>Type of Icon</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>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 <img src="image" alt="Icon" />).</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Source code for the currently selected API function is generated.</td>
</tr>
<tr>
<td><img src="image" alt="Icon" /></td>
<td>Source code for the currently selected API function is not generated.</td>
</tr>
</tbody>
</table>

(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

<table>
<thead>
<tr>
<th>Color</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Green</td>
<td>Comment</td>
</tr>
<tr>
<td>Blue</td>
<td>Reserved word for C compiler</td>
</tr>
<tr>
<td>Red</td>
<td>Numeric value</td>
</tr>
<tr>
<td>Black</td>
<td>Code section</td>
</tr>
<tr>
<td>Gray</td>
<td>File name</td>
</tr>
</tbody>
</table>

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.

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

<table>
<thead>
<tr>
<th>Message Type</th>
<th>Display Example (Default)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal message</td>
<td>ABCD abcd 0123</td>
<td>Character color Black, Background color White</td>
</tr>
<tr>
<td>Warning message</td>
<td>ABCD abcd 0123</td>
<td>Character color Blue, Background color Standard color</td>
</tr>
<tr>
<td>Error message</td>
<td>ABCD abcd 0123</td>
<td>Character color Red, Background color Light gray</td>
</tr>
</tbody>
</table>
(2) Tab selection area
Select the tab that indicates the origin of message.
The following tabs are available for the debug tool.

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

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)]

<table>
<thead>
<tr>
<th>Save Output-Tab Name</th>
<th>Saves the message corresponding to the specified tab overwriting the existing file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save Output-Tab Name As...</td>
<td>Opens the Save As dialog box for naming and saving the message corresponding to the specified tab.</td>
</tr>
</tbody>
</table>

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

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

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

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

<table>
<thead>
<tr>
<th>Pin Number</th>
<th>Configures the device pin list corresponding to the [Pin Number] tab.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macro</td>
<td>Configures the device pin list belonging to the first layer of the [Macro] tab.</td>
</tr>
<tr>
<td>Macro - Pin</td>
<td>Configures the device pin list belonging to the second layer of the [Macro] tab.</td>
</tr>
<tr>
<td>External Peripheral</td>
<td>Configures the device pin list belonging to the first layer of the [External Peripheral] tab.</td>
</tr>
<tr>
<td>External Peripheral - Pin</td>
<td>Configures the device pin list belonging to the second layer of the [External Peripheral] tab.</td>
</tr>
</tbody>
</table>
**Figure A-20. Operational Object ([Pin Number] Tab)**

![Figure A-20. Operational Object ([Pin Number] Tab)](image1)

<table>
<thead>
<tr>
<th>Pin Number</th>
<th>Pin Name</th>
<th>Function</th>
<th>I/O</th>
<th>N-ch</th>
<th>DefII</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AVREF0</td>
<td>Free</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>AVSS</td>
<td>Free</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>P10/ANCO</td>
<td>Free</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>P11/ANCI</td>
<td>Free</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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

![Figure A-21. Operational Object ([Macro] Tab: First Layer)](image2)

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

![Figure A-22. Operational Object ([Macro] Tab: Second Layer)](image3)
(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.

<table>
<thead>
<tr>
<th></th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checked</td>
<td>Displays the selected item in the device pin list.</td>
</tr>
<tr>
<td>Not checked</td>
<td>Hides the selected item in the device pin list.</td>
</tr>
</tbody>
</table>

[Function buttons]

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

<table>
<thead>
<tr>
<th>Type</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>Only character strings can be entered in the column.</td>
</tr>
<tr>
<td>Check box</td>
<td>Adds a column of check boxes.</td>
</tr>
<tr>
<td>Whole number</td>
<td>Only integers can be entered in the column.</td>
</tr>
<tr>
<td>Real number</td>
<td>Only real numbers can be entered in the column.</td>
</tr>
<tr>
<td>Date</td>
<td>Only dates in YYYYMMDD format can be entered in the column.</td>
</tr>
</tbody>
</table>

[Function buttons]

<table>
<thead>
<tr>
<th>Button</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>Adds a column that has the column heading specified in the [Name] to the right end of the device pin list.</td>
</tr>
<tr>
<td>Cancel</td>
<td>Ignores the setting and closes this dialog box.</td>
</tr>
</tbody>
</table>
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]

<table>
<thead>
<tr>
<th>Button</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make New Folder</td>
<td>Creates a &quot;New Folder&quot; below the folder selected in the Folder location.</td>
</tr>
<tr>
<td>OK</td>
<td>Specifies the folder selected in the Folder location as the destination for the files.</td>
</tr>
<tr>
<td>Cancel</td>
<td>Ignores the setting and closes this dialog box.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Excel Book (*.xls)</td>
<td>Microsoft Office Excel Book format</td>
</tr>
<tr>
<td>Bitmap (*.bmp)</td>
<td>Bitmap format</td>
</tr>
<tr>
<td>PNG (*.png)</td>
<td>PMG format</td>
</tr>
<tr>
<td>JPEG (*.jpg)</td>
<td>JPEG format</td>
</tr>
<tr>
<td>EMF (*.emf)</td>
<td>EMF format</td>
</tr>
</tbody>
</table>

**[Function buttons]**

<table>
<thead>
<tr>
<th>Button</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save</td>
<td>Outputs a file having the name specified in the <strong>[File name]</strong> and <strong>[Save as type]</strong> to the folder specified in the <strong>[Save in]</strong>.</td>
</tr>
<tr>
<td>Cancel</td>
<td>Ignores the setting and closes this dialog box.</td>
</tr>
</tbody>
</table>
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**

<table>
<thead>
<tr>
<th>Unit of Output</th>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peripheral function</td>
<td>PeripheralFunctionName.c</td>
<td>Initial function, API function</td>
</tr>
<tr>
<td></td>
<td>PeripheralFunctionName_user.c</td>
<td>Interrupt function (MD_INTxxx), callback function</td>
</tr>
<tr>
<td></td>
<td>PeripheralFunctionName.h</td>
<td>Defines macros for assigning values to registers</td>
</tr>
<tr>
<td>Project</td>
<td>CG_main.c</td>
<td>main function, R_MAIN_UserInit function</td>
</tr>
<tr>
<td></td>
<td>CG_systeminit.c</td>
<td>Call initial function of peripheral function</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Call CG_ReadResetSource</td>
</tr>
<tr>
<td></td>
<td>CG_macrodriver.h</td>
<td>Defines common macros used by all source files</td>
</tr>
<tr>
<td></td>
<td>CG_userdefine.h</td>
<td>Empty file (for user definitions)</td>
</tr>
<tr>
<td></td>
<td>CG_1kdir</td>
<td>Link directive</td>
</tr>
<tr>
<td></td>
<td>CG_start.s</td>
<td>Startup routine</td>
</tr>
<tr>
<td></td>
<td>CG_inttab.s</td>
<td>Interrupt vector table</td>
</tr>
</tbody>
</table>

### B.2 Output File

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

**Table B-2. File List (Peripheral Function)**

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>Source File Name</th>
<th>Names of API Functions Included</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>CG_system.c</td>
<td>CLOCK_Init</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CG_ChangeClockMode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CG_ChangeFrequency</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CG_SelectPowerSaveMode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CG_SelectStabTime</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CG_SelectPllMode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CG_SelectSSCGMode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WDT2_Restart</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CRC_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CRC_SetData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CRC_GetResult</td>
</tr>
<tr>
<td></td>
<td>CG_system_user.c</td>
<td>CLOCK_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CG_ReadResetSource</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_INTWDT2</td>
</tr>
<tr>
<td>External Bus</td>
<td>CG_bus.c</td>
<td>BUS_Init</td>
</tr>
<tr>
<td></td>
<td>CG_bus_user.c</td>
<td>BUS_UserInit</td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>Source File Name</td>
<td>Names of API Functions Included</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>External Bus</td>
<td>CG_bus.h</td>
<td>-</td>
</tr>
<tr>
<td>Port</td>
<td>CG_port.c</td>
<td>PORT_Init</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PORT_ChangePmnInput</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PORT_ChangePmnOutput</td>
</tr>
<tr>
<td></td>
<td>CG_port_user.c</td>
<td>PORT_UserInit</td>
</tr>
<tr>
<td></td>
<td>CG_port.h</td>
<td>-</td>
</tr>
<tr>
<td>Interrupt</td>
<td>CG_int.c</td>
<td>INTP_Init</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KEY_Init</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INT_MaskableInterruptEnable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INTPn_Disable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INTPn_Enable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KEY_Disable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KEY_Enable</td>
</tr>
<tr>
<td></td>
<td>CG_int_user.c</td>
<td>INTP_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KEY_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_INTNMI</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_INTPn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_INTKR</td>
</tr>
<tr>
<td></td>
<td>CG_int.h</td>
<td>-</td>
</tr>
<tr>
<td>Serial</td>
<td>CG_serial.c</td>
<td>UARTAn_Init</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UARTAn_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UARTAn_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UARTAn_SendData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UARTAn_ReceiveData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UARTBn_Init</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UARTBn_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UARTBn_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UARTBn_SendData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UARTBn_ReceiveData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UARTCn_Init</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UARTCn_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UARTCn_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UARTCn_SendData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UARTCn_ReceiveData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSIEn_Init</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSIEn_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSIEn_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSIEn_SendData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSIEn_ReceiveData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSIEn_SendReceiveData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSIEn_SendReceiveData</td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>Source File Name</td>
<td>Names of API Functions Included</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| 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  
|                     |                       | UARTBn_UserInit  
|                     |                       | UARTBn_SendEndCallback  
|                     |                       | UARTBn_ReceiveEndCallback  
|                     |                       | UARTBn_SingleErrorCallback  
|                     |                       | UARTBn_FIFOErrorCallback  
|                     |                       | UARTBn_TimeoutErrorCallback  
|                     |                       | UARTBn_SoftOverRunCallback  
|                     |                       | UARTCn_UserInit  
|                     |                       | UARTCn_SendEndCallback  
|                     |                       | UARTCn_ReceiveEndCallback  
|                     |                       | UARTCn_ErrorCallback  
|                     |                       | UARTCn_SoftOverRunCallback  
|                     |                       | 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  
<p>|                     |                       | IIC0n_SlaveErrorCallback  |</p>
<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>Source File Name</th>
<th>Names of API Functions Included</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial</td>
<td>CG_serial_user.c</td>
<td>IIC0n_GetStopConditionCallback</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_INTUA\nT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_INTUA\nR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_INTUB\nTIT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_INTUB\nTIF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_INTUB\nTIR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_INTUB\nTIRE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_INTUB\nTITO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_INTUC\nT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_INTUC\nR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_INTCB\nT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_INTCB\nR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_INTCE\nT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_INTCE\nTIOF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_INTCF\nT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_INTCF\nR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_INTIC\n</td>
</tr>
<tr>
<td></td>
<td>CG_serial.h</td>
<td>-</td>
</tr>
<tr>
<td>A/D Converter</td>
<td>CG_ad.c</td>
<td>AD_Init</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AD_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AD_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AD_SelectADChannel</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AD_SetPFTCondition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AD_Read</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AD_ReadByte</td>
</tr>
<tr>
<td></td>
<td>CG_ad_user.c</td>
<td>AD_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_INTAD</td>
</tr>
<tr>
<td></td>
<td>CG_ad.h</td>
<td>-</td>
</tr>
<tr>
<td>D/A Converter</td>
<td>CG_da.c</td>
<td>DAn_Init</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DAn_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DAn_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DAn_SetValue</td>
</tr>
<tr>
<td></td>
<td>CG_da_user.c</td>
<td>DAn_UserInit</td>
</tr>
<tr>
<td></td>
<td>CG_da.h</td>
<td>-</td>
</tr>
<tr>
<td>Timer</td>
<td>CG_timer.c</td>
<td>TMPn_Init</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMPn_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMPn_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMPn_ChangeTimerCondition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMPn_GetPulseWidth</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMPn_GetFreeRunningValue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMPn_ChangeDuty</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMPn_SoftwareTriggerOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMQ0_Init</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMQ0_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMQ0_Stop</td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>Source File Name</td>
<td>Names of API Functions Included</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>Timer</td>
<td>CG_timer.c</td>
<td>TMQ0_ChangeTimerCondition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMQ0_GetPulseWidth</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMQ0_GetFreeRunningValue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMQ0_ChangeDuty</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMQ0_SoftwareTriggerOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TAAn_Init</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TAAn_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TAAn_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TAAn_ChangeTimerCondition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TAAn_ControlOutputToggle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TAAn_GetPulseWidth</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TAAn_GetFreeRunningValue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TAAn_ChangeDuty</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TAAn_SoftwareTriggerOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TABn_Init</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TABn_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TABn_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TABn_ChangeTimerCondition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TABn_ControlOutputToggle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TABn_GetPulseWidth</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TABn_GetFreeRunningValue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TABn_ChangeDuty</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TABn_SoftwareTriggerOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMT0_Init</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMT0_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMT0_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMT0_ChangeTimerCondition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMT0_GetPulseWidth</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMT0_GetFreeRunningValue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMT0_ChangeDuty</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMT0_SoftwareTriggerOn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMMn_Init</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMMn_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMMn_Stop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMMn_ChangeTimerCondition</td>
</tr>
<tr>
<td>CG_timer_user.c</td>
<td></td>
<td>TMPn_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMQ0_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TAAn_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TABn_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMT0_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMMn_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_INTTPnOV</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_INTTPnCCm</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_INTTQnOV</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_INTTQnCCm</td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>Source File Name</td>
<td>Names of API Functions Included</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>Timer</td>
<td>CG_timer_user.c</td>
<td>MD_INTTAAOV&lt;br&gt;MD_INTTAAOCm&lt;br&gt;MD_INTTABnOV&lt;br&gt;MD_INTTABnCCm&lt;br&gt;MD_INTTT0EC&lt;br&gt;MD_INTTT0OV&lt;br&gt;MD_INTTT0CCm&lt;br&gt;MD_INTTMnEQ0</td>
</tr>
<tr>
<td></td>
<td>CG_timer.h</td>
<td>-</td>
</tr>
<tr>
<td>Watch Timer</td>
<td>CG_wt.c</td>
<td>WT_Init&lt;br&gt;WT_Start&lt;br&gt;WT_Stop</td>
</tr>
<tr>
<td></td>
<td>CG_wt_user.c</td>
<td>WT_UserInit&lt;br&gt;MD_INTWT&lt;br&gt;MD_INTWTI</td>
</tr>
<tr>
<td></td>
<td>CG_wt.h</td>
<td>-</td>
</tr>
<tr>
<td>Real-time Clock</td>
<td>CG_rtc.c</td>
<td>RTC_Init&lt;br&gt;RTC.CounterEnable&lt;br&gt;RTC.CounterDisable&lt;br&gt;RTC.SetHourSystem&lt;br&gt;RTC.CounterSet&lt;br&gt;RTC.CounterGet&lt;br&gt;RTC.ConstPeriodInterruptEnable&lt;br&gt;RTC.ConstPeriodInterruptDisable&lt;br&gt;RTC.AlarmEnable&lt;br&gt;RTC.AlarmDisable&lt;br&gt;RTC.AlarmSet&lt;br&gt;RTC.AlarmGet&lt;br&gt;RTC.IntervalStart&lt;br&gt;RTC.IntervalStop&lt;br&gt;RTC.IntervalInterruptEnable&lt;br&gt;RTC.IntervalInterruptDisable&lt;br&gt;RTC_RC1CK1HZ_OutputEnable&lt;br&gt;RTC_RC1CK1HZ_OutputDisable&lt;br&gt;RTC_RC1CKO_OutputEnable&lt;br&gt;RTC_RC1CKO_OutputDisable&lt;br&gt;RTC_RC1CKDIV_OutputEnable&lt;br&gt;RTC_RC1CKDIV_OutputDisable&lt;br&gt;RTC_RTC1HZ_OutputEnable&lt;br&gt;RTC_RTC1HZ_OutputDisable&lt;br&gt;RTC_RTCCL_OutputEnable&lt;br&gt;RTC_RTCCL_OutputDisable&lt;br&gt;RTC_RTCDIV_OutputEnable&lt;br&gt;RTC_RTCDIV_OutputDisable&lt;br&gt;RTC_ChangeCorrectionValue</td>
</tr>
<tr>
<td></td>
<td>CG_rtc_user.c</td>
<td>RTC_UserInit</td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>Source File Name</td>
<td>Names of API Functions Included</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>Real-time Clock</td>
<td>CG rtc_user.c</td>
<td>MD_INTRRTCn</td>
</tr>
<tr>
<td></td>
<td>CG rtc.h</td>
<td>-</td>
</tr>
<tr>
<td>Real-Time Output</td>
<td>CG rto.c</td>
<td>RTOn_Init</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RTOn_Enable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RTOn_Disable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RTOn_Set2BitData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RTOn_Set4BitData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RTOn_Set6BitData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RTOn_Set8BitData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RTOn_SetHigh2BitData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RTOn_SetLow2BitData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RTOn_SetHigh4BitData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RTOn_SetLow4BitData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RTOn_GetValue</td>
</tr>
<tr>
<td></td>
<td>CG rto_user.c</td>
<td>RTOn_UserInit</td>
</tr>
<tr>
<td></td>
<td>CG rto.h</td>
<td>-</td>
</tr>
<tr>
<td>DMA</td>
<td>CG dma.c</td>
<td>DMAn_Init</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DMAn_Enable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DMAn_Disable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DMAn_CheckStatus</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DMAn_SetData</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DMAn_SoftwareTriggerOn</td>
</tr>
<tr>
<td></td>
<td>CG dma_user.c</td>
<td>DMAn_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_INTDMAAn</td>
</tr>
<tr>
<td></td>
<td>CG dma.h</td>
<td>-</td>
</tr>
<tr>
<td>LVI</td>
<td>CG lvi.c</td>
<td>LVI_Init</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LVI_InterruptModeStart</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LVI_ResetModeStart</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LVI_Start</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LVI_Stop</td>
</tr>
<tr>
<td></td>
<td>CG lvi_user.c</td>
<td>LVI_UserInit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_INTLVI</td>
</tr>
<tr>
<td></td>
<td>CG lvi.h</td>
<td>-</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Peripheral Function</th>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>CLOCK_Init</td>
<td>Performs initialization necessary to control clock functions.</td>
</tr>
<tr>
<td></td>
<td>CLOCK_UserInit</td>
<td>Performs user-defined initialization relating to the clock.</td>
</tr>
<tr>
<td></td>
<td>CG_ReadResetSource</td>
<td>Performs processing in response to a reset signal.</td>
</tr>
<tr>
<td></td>
<td>CG_ChangeClockMode</td>
<td>Changes the CPU clock.</td>
</tr>
<tr>
<td></td>
<td>CG_ChangeFrequency</td>
<td>Changes the CPU clock division ratio.</td>
</tr>
<tr>
<td></td>
<td>CG_SelectPowerSaveMode</td>
<td>Configures the CPU's standby function.</td>
</tr>
<tr>
<td></td>
<td>CG_SelectStabTime</td>
<td>Selects the oscillation stabilization time for the X1 oscillator.</td>
</tr>
<tr>
<td></td>
<td>CG_SelectPllMode</td>
<td>Selects the operation mode of the PLL function.</td>
</tr>
<tr>
<td></td>
<td>CG_SelectSSCGMode</td>
<td>Selects the operation mode of the SSCG (Spread Spectrum Clock Generator).</td>
</tr>
<tr>
<td></td>
<td>WDT2_Restart</td>
<td>Clears the watchdog timer counter and resumes counting.</td>
</tr>
<tr>
<td></td>
<td>CRC_Start</td>
<td>Begins detection of data-block errors.</td>
</tr>
<tr>
<td></td>
<td>CRC_SetData</td>
<td>Sets data in the CRC input register (CRCIN).</td>
</tr>
<tr>
<td></td>
<td>CRC_GetResult</td>
<td>Reads the results of the calculation stored in the CRC data register (CRCD).</td>
</tr>
<tr>
<td>External Bus</td>
<td>BUS_Init</td>
<td>Performs initialization necessary to control external bus interface functions (functions to connect an external bus to areas other than onboard ROM, ROM and RAM).</td>
</tr>
<tr>
<td></td>
<td>BUS_UserInit</td>
<td>Performs user-defined initialization relating to the external bus interface.</td>
</tr>
<tr>
<td>Port</td>
<td>PORT_Init</td>
<td>Performs initialization necessary to control port functions.</td>
</tr>
<tr>
<td></td>
<td>PORT_UserInit</td>
<td>Performs user-defined initialization relating to the port.</td>
</tr>
<tr>
<td></td>
<td>PORT_ChangePmnInput</td>
<td>Switches the pin’s I/O mode from output mode to input mode.</td>
</tr>
<tr>
<td></td>
<td>PORT_ChangePmnOutput</td>
<td>Switches the pin’s I/O mode from input mode to output mode.</td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>API Function Name</td>
<td>Function</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Interrupt</td>
<td>INTP_Init</td>
<td>Performs initialization necessary to control the external interrupt INTP functions.</td>
</tr>
<tr>
<td></td>
<td>INTP_UserInit</td>
<td>Performs user-defined initialization relating to the external interrupt INTP functions.</td>
</tr>
<tr>
<td></td>
<td>KEY_Init</td>
<td>Performs initialization necessary to control the key interrupt INTKR functions.</td>
</tr>
<tr>
<td></td>
<td>KEY_UserInit</td>
<td>Performs user-defined initialization relating to the key interrupt INTKR functions.</td>
</tr>
<tr>
<td></td>
<td>INT_MaskableInterruptEnable</td>
<td>Disables/enables the acceptance of the maskable interrupts.</td>
</tr>
<tr>
<td></td>
<td>INTPn_Disable</td>
<td>Disables the acceptance of the maskable interrupts INTPn (external interrupt requests).</td>
</tr>
<tr>
<td></td>
<td>INTPn_Enable</td>
<td>Enables the acceptance of the maskable interrupts INTPn (external interrupt requests).</td>
</tr>
<tr>
<td></td>
<td>KEY_Disable</td>
<td>Disables the acceptance of the key interrupts INTKR.</td>
</tr>
<tr>
<td></td>
<td>KEY_Enable</td>
<td>Enables the acceptance of the key interrupts INTKR.</td>
</tr>
<tr>
<td>Serial</td>
<td>UARTAn_Init</td>
<td>Performs initialization necessary to control the asynchronous serial interface A (UARTA) functions.</td>
</tr>
<tr>
<td></td>
<td>UARTAn_UserInit</td>
<td>Performs user-defined initialization relating to the asynchronous serial interface A (UARTA).</td>
</tr>
<tr>
<td></td>
<td>UARTAn_Start</td>
<td>Enables asynchronous serial interface A (UARTA).</td>
</tr>
<tr>
<td></td>
<td>UARTAn_Stop</td>
<td>Disables asynchronous serial interface A (UARTA).</td>
</tr>
<tr>
<td></td>
<td>UARTAn_SendData</td>
<td>Starts UARTAn data transmission.</td>
</tr>
<tr>
<td></td>
<td>UARTAn_ReceiveData</td>
<td>Starts UARTAn data reception.</td>
</tr>
<tr>
<td></td>
<td>UARTAn_SendEndCallback</td>
<td>Performs processing in response to the UARTAn consecutive transmission enable interrupt INTUAiT.</td>
</tr>
<tr>
<td></td>
<td>UARTAn_ReceiveEndCallback</td>
<td>Performs processing in response to the UARTAn reception completion interrupt INTUAIR.</td>
</tr>
<tr>
<td></td>
<td>UARTAn_ErrorCallback</td>
<td>Performs processing in response to the UARTAn reception error interrupt INTUAIR (overrun error, framing error, parity error).</td>
</tr>
<tr>
<td></td>
<td>UARTAn_SoftOverRunCallback</td>
<td>Performs processing in response to detection of overrun error.</td>
</tr>
<tr>
<td></td>
<td>UARTBn_Init</td>
<td>Performs initialization necessary to control the asynchronous serial interface B (UARTB) functions.</td>
</tr>
<tr>
<td></td>
<td>UARTBn_UserInit</td>
<td>Performs user-defined initialization relating to the asynchronous serial interface B (UARTB).</td>
</tr>
<tr>
<td></td>
<td>UARTBn_Start</td>
<td>Enables asynchronous serial interface B (UARTB).</td>
</tr>
<tr>
<td></td>
<td>UARTBn_Stop</td>
<td>Disables asynchronous serial interface B (UARTB).</td>
</tr>
<tr>
<td></td>
<td>UARTBn_SendData</td>
<td>Starts UARTBn data transmission.</td>
</tr>
<tr>
<td></td>
<td>UARTBn_ReceiveData</td>
<td>Starts UARTBn data reception.</td>
</tr>
<tr>
<td></td>
<td>UARTBn_SendEndCallback</td>
<td>Performs processing consequent to the transmission enable interrupt INTUBnITIT and the FIFO transmission completion interrupt INTUBnITIF.</td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>API Function Name</td>
<td>Function</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Serial</td>
<td>UARTBn_ReceiveEndCallback</td>
<td>Performs processing in response to the reception completion interrupt INTUBnTIR.</td>
</tr>
<tr>
<td></td>
<td>UARTBn_SingleErrorCallback</td>
<td>Performs processing in response to the reception error interrupt INTUBnTIRE (overrun error, framing error, parity error).</td>
</tr>
<tr>
<td></td>
<td>UARTBn_FIFOErrorCallback</td>
<td>Performs processing in response to the reception error interrupt INTUBnTIRE (overrun error, framing error, parity error).</td>
</tr>
<tr>
<td></td>
<td>UARTBn_TimeoutErrorCallback</td>
<td>Performs processing in response to the reception timeout error interrupt INTUBnTITO.</td>
</tr>
<tr>
<td></td>
<td>UARTBn_SoftOverRunCallback</td>
<td>Performs processing in response to detection of overrun error.</td>
</tr>
<tr>
<td>UARTCn_Init</td>
<td>Performs initialization necessary to control the asynchronous serial interface A (UARTA) functions.</td>
<td></td>
</tr>
<tr>
<td>UARTCn_UserInit</td>
<td>Performs user-defined initialization relating to the asynchronous serial interface A (UARTA).</td>
<td></td>
</tr>
<tr>
<td>UARTCn_Start</td>
<td>Enables asynchronous serial interface A (UARTA).</td>
<td></td>
</tr>
<tr>
<td>UARTCn_Stop</td>
<td>Disables asynchronous serial interface A (UARTA).</td>
<td></td>
</tr>
<tr>
<td>UARTCn_SendData</td>
<td>Starts UARTAn data transmission.</td>
<td></td>
</tr>
<tr>
<td>UARTCn_ReceiveData</td>
<td>Starts UARTAn data reception.</td>
<td></td>
</tr>
<tr>
<td>UARTCn_SendEndCallback</td>
<td>Performs processing in response to the UARTAn consecutive transmission enable interrupt INTUAiT.</td>
<td></td>
</tr>
<tr>
<td>UARTCn_ReceiveEndCallback</td>
<td>Performs processing in response to the UARTAn reception completion interrupt INTUAinR.</td>
<td></td>
</tr>
<tr>
<td>UARTCn_ErrorCallback</td>
<td>Performs processing in response to the UARTAn reception error interrupt INTUAinR (overrun error, framing error, parity error).</td>
<td></td>
</tr>
<tr>
<td>UARTCn_SoftOverRunCallback</td>
<td>Performs processing in response to detection of overrun error.</td>
<td></td>
</tr>
<tr>
<td>CSIBn_Init</td>
<td>Performs initialization necessary to control the 3-wire variable-length serial I/O B (CSIB) functions.</td>
<td></td>
</tr>
<tr>
<td>CSIBn_UserInit</td>
<td>Performs user-defined initialization relating to the 3-wire variable-length serial I/O B (CSIB).</td>
<td></td>
</tr>
<tr>
<td>CSIBn_Start</td>
<td>Enables 3-wire variable-length serial I/O B (CSIB).</td>
<td></td>
</tr>
<tr>
<td>CSIBn_Stop</td>
<td>Disables 3-wire variable-length serial I/O B (CSIB).</td>
<td></td>
</tr>
<tr>
<td>CSIBn_SendData</td>
<td>Starts CSIB data transmission.</td>
<td></td>
</tr>
<tr>
<td>CSIBn_ReceiveData</td>
<td>Starts CSIB data reception.</td>
<td></td>
</tr>
<tr>
<td>CSIBn_SendReceiveData</td>
<td>Performs processing in response to the CSIBn reception completion interrupt INTCBnR or the CSIBn consecutive transmission write enable interrupt INTCBnT.</td>
<td></td>
</tr>
<tr>
<td>CSIBn_ReceiveEndCallback</td>
<td>Performs processing in response to the CSIBn reception completion interrupt INTCBnR.</td>
<td></td>
</tr>
<tr>
<td>CSIBn_ErrorCallback</td>
<td>Performs processing in response to the CSIBn reception error interrupt INTCBnR (overrun error).</td>
<td></td>
</tr>
<tr>
<td>CSIEn_Init</td>
<td>Performs initialization necessary to control the 3-wire variable-length serial I/O E (CSIE) functions.</td>
<td></td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>API Function Name</td>
<td>Function</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Serial</td>
<td>CSIEn_UserInit</td>
<td>Performs user-defined initialization relating to the 3-wire variable-length serial I/O E (CSIE).</td>
</tr>
<tr>
<td>CSIEn_Start</td>
<td></td>
<td>Enables 3-wire variable-length serial I/O E (CSIE).</td>
</tr>
<tr>
<td>CSIEn_Stop</td>
<td></td>
<td>Disables 3-wire variable-length serial I/O E (CSIE).</td>
</tr>
<tr>
<td>CSIEn_SendData</td>
<td></td>
<td>Starts CSIE data transmission.</td>
</tr>
<tr>
<td>CSIEn_ReceiveData</td>
<td></td>
<td>Starts CSIE data reception.</td>
</tr>
<tr>
<td>CSIEn_SendReceiveData</td>
<td></td>
<td>Starts CSIE data transmission/reception.</td>
</tr>
<tr>
<td>CSIEn_SendEndCallback</td>
<td></td>
<td>Performs processing in response to the CSIEn transmission/reception completion interrupt INTCEn.</td>
</tr>
<tr>
<td>CSIEn_ReceiveEndCallback</td>
<td></td>
<td>Performs processing in response to the CSIEn transmission/reception completion interrupt INTCEn.</td>
</tr>
<tr>
<td>CSIEn_ErrorCallback</td>
<td></td>
<td>Performs processing in response to the CSIEnBUF overflow interrupt INTCEnTIoF.</td>
</tr>
<tr>
<td>CSIFn_Init</td>
<td></td>
<td>Performs initialization necessary to control the 3-wire variable-length serial I/O F (CSIF).</td>
</tr>
<tr>
<td>CSIFn_UserInit</td>
<td></td>
<td>Performs user-defined initialization relating to the 3-wire variable-length serial I/O F (CSIF).</td>
</tr>
<tr>
<td>CSIFn_Start</td>
<td></td>
<td>Enables 3-wire variable-length serial I/O F (CSIF).</td>
</tr>
<tr>
<td>CSIFn_Stop</td>
<td></td>
<td>Disables 3-wire variable-length serial I/O F (CSIF).</td>
</tr>
<tr>
<td>CSIFn_SendData</td>
<td></td>
<td>Starts CSIF data transmission.</td>
</tr>
<tr>
<td>CSIFn_ReceiveData</td>
<td></td>
<td>Starts CSIF data reception.</td>
</tr>
<tr>
<td>CSIFn_SendReceiveData</td>
<td></td>
<td>Starts CSIF data transmission/reception.</td>
</tr>
<tr>
<td>CSIFn_SendEndCallback</td>
<td></td>
<td>Performs processing in response to the CSIFn transmission/reception completion interrupt INTCFn.</td>
</tr>
<tr>
<td>CSIFn_ReceiveEndCallback</td>
<td></td>
<td>Performs processing in response to the CSIFn transmission/reception completion interrupt INTCFn.</td>
</tr>
<tr>
<td>CSIFn_ErrorCallback</td>
<td></td>
<td>Performs processing in response to the CSIFn reception error interrupt INTCFnR (overrun error).</td>
</tr>
<tr>
<td>IIC0n_Init</td>
<td></td>
<td>Performs initialization necessary to control the IIC bus functions.</td>
</tr>
<tr>
<td>IIC0n_UserInit</td>
<td></td>
<td>Performs user-defined initialization relating to the IIC bus.</td>
</tr>
<tr>
<td>IIC0n_Stop</td>
<td></td>
<td>Ends IIC0n communication.</td>
</tr>
<tr>
<td>IIC0n_StopCondition</td>
<td></td>
<td>Generates a stop condition.</td>
</tr>
<tr>
<td>IIC0n_MasterSendStart</td>
<td></td>
<td>Starts IIC0n master transmission.</td>
</tr>
<tr>
<td>IIC0n_MasterReceiveStart</td>
<td></td>
<td>Starts IIC0n master reception.</td>
</tr>
<tr>
<td>IIC0n_MasterSendEndCallback</td>
<td></td>
<td>Performs processing in response to the IICn master transfer completion interrupt INTIICn.</td>
</tr>
<tr>
<td>IIC0n_MasterReceiveEndCallback</td>
<td></td>
<td>Performs processing in response to the IICn master transfer completion interrupt INTIICn.</td>
</tr>
<tr>
<td>IIC0n_MasterErrorCallback</td>
<td></td>
<td>Performs processing in response to detection of error in IICn master communication.</td>
</tr>
<tr>
<td>IIC0n_SlaveSendStart</td>
<td></td>
<td>Starts IIC0n slave transmission.</td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>API Function Name</td>
<td>Function</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Serial</td>
<td>IIC0n_SlaveReceiveStart</td>
<td>Starts IIC0n slave reception.</td>
</tr>
<tr>
<td></td>
<td>IIC0n_SlaveSendEndCallback</td>
<td>Performs processing in response to the IICn slave transfer completion interrupt INTIICn.</td>
</tr>
<tr>
<td></td>
<td>IIC0n_SlaveReceiveEndCallback</td>
<td>Performs processing in response to the IICn slave transfer completion interrupt INTIICn.</td>
</tr>
<tr>
<td></td>
<td>IIC0n_SlaveErrorCallback</td>
<td>Performs processing in response to detection of error in IICn slave communication.</td>
</tr>
<tr>
<td></td>
<td>IIC0n_GetStopConditionCallback</td>
<td>Performs processing in response to detection of stop condition.</td>
</tr>
<tr>
<td>A/D Converter</td>
<td>AD_Init</td>
<td>Performs initialization necessary to control A/D converter functions.</td>
</tr>
<tr>
<td></td>
<td>AD_UserInit</td>
<td>Performs user-defined initialization relating to the A/D converter.</td>
</tr>
<tr>
<td></td>
<td>AD_Start</td>
<td>Starts A/D conversion.</td>
</tr>
<tr>
<td></td>
<td>AD_Stop</td>
<td>Ends A/D conversion.</td>
</tr>
<tr>
<td></td>
<td>AD_SelectADChannel</td>
<td>Configures the analog voltage input pin for A/D conversion.</td>
</tr>
<tr>
<td></td>
<td>AD_SetPFTCondition</td>
<td>Sets the information for operation in power-fail compare mode (comparison value and A/D conversion end interrupt INTAD trigger).</td>
</tr>
<tr>
<td></td>
<td>AD_Read</td>
<td>Reads the results of A/D conversion (10 bits).</td>
</tr>
<tr>
<td></td>
<td>AD_ReadByte</td>
<td>Reads the results of A/D conversion (8 bits; most significant 8 bits of 10-bit resolution).</td>
</tr>
<tr>
<td>D/A Converter</td>
<td>DAn_Init</td>
<td>Performs initialization necessary to control D/A converter functions.</td>
</tr>
<tr>
<td></td>
<td>DAn_UserInit</td>
<td>Performs user-defined initialization relating to the D/A converter.</td>
</tr>
<tr>
<td></td>
<td>DAn_Start</td>
<td>Starts D/A conversion.</td>
</tr>
<tr>
<td></td>
<td>DAn_Stop</td>
<td>Ends D/A conversion.</td>
</tr>
<tr>
<td></td>
<td>DAn_SetValue</td>
<td>Sets the analog voltage output to the ANOn pin.</td>
</tr>
<tr>
<td>Timer</td>
<td>TMPn_Init</td>
<td>Performs initialization necessary to control 16-bit timer/event counter P (TMP) functions.</td>
</tr>
<tr>
<td></td>
<td>TMPn_UserInit</td>
<td>Performs user-defined initialization relating to the 16-bit timer/event counter P (TMP).</td>
</tr>
<tr>
<td></td>
<td>TMPn_Start</td>
<td>Starts the count for 16-bit timer/event counter P (TMP).</td>
</tr>
<tr>
<td></td>
<td>TMPn_Stop</td>
<td>Ends the count for 16-bit timer/event counter P (TMP).</td>
</tr>
<tr>
<td></td>
<td>TMPn_ChangeTimerCondition</td>
<td>Changes the counter value for 16-bit timer/event counter P (TMP).</td>
</tr>
<tr>
<td></td>
<td>TMPn_GetPulseWidth</td>
<td>Reads the pulse width of 16-bit timer/event counter P (TMP) (high/low level width).</td>
</tr>
<tr>
<td></td>
<td>TMPn_GetFreeRunningValue</td>
<td>Reads the value captured by 16-bit timer/event counter P (TMP).</td>
</tr>
<tr>
<td></td>
<td>TMPn_ChangeDuty</td>
<td>Changes the duty ratio of the PWM signal.</td>
</tr>
<tr>
<td></td>
<td>TMPn_SoftwareTriggerOn</td>
<td>Generates the trigger (software trigger) for timer output.</td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>API Function Name</td>
<td>Function</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Timer</td>
<td>TMQ0_Init</td>
<td>Performs initialization necessary to control 16-bit timer/event counter Q (TMQ) functions.</td>
</tr>
<tr>
<td></td>
<td>TMQ0_UserInit</td>
<td>Performs user-defined initialization relating to the 16-bit timer/event counter Q (TMQ).</td>
</tr>
<tr>
<td></td>
<td>TMQ0_Start</td>
<td>Starts the count for 16-bit timer/event counter Q (TMQ).</td>
</tr>
<tr>
<td></td>
<td>TMQ0_Stop</td>
<td>Ends the count for 16-bit timer/event counter Q (TMQ).</td>
</tr>
<tr>
<td></td>
<td>TMQ0_ChangeTimerCondition</td>
<td>Changes the counter value for 16-bit timer/event counter Q (TMQ).</td>
</tr>
<tr>
<td></td>
<td>TMQ0_GetPulseWidth</td>
<td>Reads the pulse width of 16-bit timer/event counter Q (TMQ) (high/low level width).</td>
</tr>
<tr>
<td></td>
<td>TMQ0_GetFreeRunningValue</td>
<td>Reads the value captured by 16-bit timer/event counter Q (TMQ).</td>
</tr>
<tr>
<td></td>
<td>TMQ0_ChangeDuty</td>
<td>Changes the duty ratio of the PWM signal.</td>
</tr>
<tr>
<td></td>
<td>TMQ0_SoftwareTriggerOn</td>
<td>Generates the trigger (software trigger) for timer output.</td>
</tr>
<tr>
<td></td>
<td>TAAn_Init</td>
<td>Performs initialization necessary to control 16-bit timer/event counter AA (TAA) functions.</td>
</tr>
<tr>
<td></td>
<td>TAAn_UserInit</td>
<td>Performs user-defined initialization relating to the 16-bit timer/event counter AA (TAA).</td>
</tr>
<tr>
<td></td>
<td>TAAn_Start</td>
<td>Starts the count for 16-bit timer/event counter AA (TAA).</td>
</tr>
<tr>
<td></td>
<td>TAAn_Stop</td>
<td>Ends the count for 16-bit timer/event counter AA (TAA).</td>
</tr>
<tr>
<td></td>
<td>TAAn_ChangeTimerCondition</td>
<td>Changes the counter value for 16-bit timer/event counter AA (TAA).</td>
</tr>
<tr>
<td></td>
<td>TAAn_ControlOutputToggle</td>
<td>Changes the toggle control of 16-bit timer/event counter AA (TAA).</td>
</tr>
<tr>
<td></td>
<td>TAAn_GetPulseWidth</td>
<td>Reads the pulse width of 16-bit timer/event counter AA (TAA) (high/low level width).</td>
</tr>
<tr>
<td></td>
<td>TAAn_GetFreeRunningValue</td>
<td>Reads the value captured by 16-bit timer/event counter AA (TAA).</td>
</tr>
<tr>
<td></td>
<td>TAAn_ChangeDuty</td>
<td>Changes the duty ratio of the PWM signal.</td>
</tr>
<tr>
<td></td>
<td>TAAn_SoftwareTriggerOn</td>
<td>Generates the trigger (software trigger) for timer output.</td>
</tr>
<tr>
<td></td>
<td>TABn_Init</td>
<td>Performs initialization necessary to control 16-bit timer/event counter AB (TAB) functions.</td>
</tr>
<tr>
<td></td>
<td>TABn_UserInit</td>
<td>Performs user-defined initialization relating to the 16-bit timer/event counter AB (TAB).</td>
</tr>
<tr>
<td></td>
<td>TABn_Start</td>
<td>Starts the count for 16-bit timer/event counter AB (TAB).</td>
</tr>
<tr>
<td></td>
<td>TABn_Stop</td>
<td>Ends the count for 16-bit timer/event counter AB (TAB).</td>
</tr>
<tr>
<td></td>
<td>TABn_ChangeTimerCondition</td>
<td>Changes the counter value for 16-bit timer/event counter AB (TAB).</td>
</tr>
<tr>
<td></td>
<td>TABn_ControlOutputToggle</td>
<td>Changes the toggle control of 16-bit timer/event counter AB (TAB).</td>
</tr>
<tr>
<td></td>
<td>TABn_GetPulseWidth</td>
<td>Reads the pulse width of 16-bit timer/event counter AB (TAB) (high/low level width).</td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>API Function Name</td>
<td>Function</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Timer</td>
<td>TABn_GetFreeRunningValue</td>
<td>Reads the value captured by 16-bit timer/event counter AB (TAB).</td>
</tr>
<tr>
<td></td>
<td>TABn_ChangeDuty</td>
<td>Changes the duty ratio of the PWM signal.</td>
</tr>
<tr>
<td></td>
<td>TABn_SoftwareTriggerOn</td>
<td>Generates the trigger (software trigger) for timer output.</td>
</tr>
<tr>
<td></td>
<td>TMT0_Init</td>
<td>Performs initialization necessary to control 16-bit timer/event counter T (TMT).</td>
</tr>
<tr>
<td></td>
<td>TMT0_UserInit</td>
<td>Performs user-defined initialization relating to the 16-bit timer/event counter T (TMT).</td>
</tr>
<tr>
<td></td>
<td>TMT0_Start</td>
<td>Starts the count for 16-bit timer/event counter T (TMT).</td>
</tr>
<tr>
<td></td>
<td>TMT0_Stop</td>
<td>Ends the count for 16-bit timer/event counter T (TMT).</td>
</tr>
<tr>
<td></td>
<td>TMT0_ChangeTimerCondition</td>
<td>Changes the counter value for 16-bit timer/event counter T (TMT).</td>
</tr>
<tr>
<td></td>
<td>TMT0_GetPulseWidth</td>
<td>Reads the pulse width of 16-bit timer/event counter T (TMT) (high/low level width).</td>
</tr>
<tr>
<td></td>
<td>TMT0_GetFreeRunningValue</td>
<td>Reads the value captured by 16-bit timer/event counter T (TMT).</td>
</tr>
<tr>
<td></td>
<td>TMT0_ChangeDuty</td>
<td>Changes the duty ratio of the PWM signal.</td>
</tr>
<tr>
<td></td>
<td>TMT0_SoftwareTriggerOn</td>
<td>Generates the trigger (software trigger) for timer output.</td>
</tr>
<tr>
<td></td>
<td>TMT0_EnableHold</td>
<td>Changes the encoder counter control of 16-bit timer/event counter T (TMT) to holding count value.</td>
</tr>
<tr>
<td></td>
<td>TMT0_DisableHold</td>
<td>Changes the encoder counter control of 16-bit timer/event counter T (TMT) to normal operation.</td>
</tr>
<tr>
<td></td>
<td>TMT0_ChangeCountValue</td>
<td>Changes the initial counter value for 16-bit timer/event counter T (TMT).</td>
</tr>
<tr>
<td></td>
<td>TMMn_Init</td>
<td>Performs initialization necessary to control 16-bit interval timer M (TMM).</td>
</tr>
<tr>
<td></td>
<td>TMMn_UserInit</td>
<td>Performs user-defined initialization relating to the 16-bit interval timer M (TMM).</td>
</tr>
<tr>
<td></td>
<td>TMMn_Start</td>
<td>Starts the count for 16-bit interval timer M (TMM).</td>
</tr>
<tr>
<td></td>
<td>TMMn_Stop</td>
<td>Ends the count for 16-bit interval timer M (TMM).</td>
</tr>
<tr>
<td></td>
<td>TMMn_ChangeTimerCondition</td>
<td>Changes the counter value for 16-bit interval timer M (TMM).</td>
</tr>
<tr>
<td>Watch Timer</td>
<td>WT_Init</td>
<td>Performs initialization necessary to control watch timer functions.</td>
</tr>
<tr>
<td></td>
<td>WT_UserInit</td>
<td>Performs user-defined initialization relating to the watch timer.</td>
</tr>
<tr>
<td></td>
<td>WT_Start</td>
<td>Clears the watch timer counter and resumes counting.</td>
</tr>
<tr>
<td></td>
<td>WT_Stop</td>
<td>Ends the count for watch timer.</td>
</tr>
<tr>
<td>Real-time Clock</td>
<td>RTC_Init</td>
<td>Performs initialization necessary to control real-time counter functions.</td>
</tr>
<tr>
<td></td>
<td>RTC_UserInit</td>
<td>Performs user-defined initialization relating to the real-time counter.</td>
</tr>
<tr>
<td></td>
<td>RTC_CounterEnable</td>
<td>Starts the count of the real-time counter (year, month, weekday, day, hour, minute, second).</td>
</tr>
<tr>
<td>Peripheral Function</td>
<td>API Function Name</td>
<td>Function</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Real-time Clock</td>
<td>RTC_CounterDisable</td>
<td>Ends the count of the real-time counter (year, month, weekday, day, hour, minute, second).</td>
</tr>
<tr>
<td></td>
<td>RTC_SetHourSystem</td>
<td>Sets the clock type (12-hour or 24-hour clock) of the real-time counter.</td>
</tr>
<tr>
<td></td>
<td>RTC.CounterSet</td>
<td>Sets the counter value (year, month, weekday, day, hour, minute, second) of the real-time counter.</td>
</tr>
<tr>
<td></td>
<td>RTC.CounterGet</td>
<td>Reads the counter value (year, month, weekday, day, hour, minute, second) of the real-time counter.</td>
</tr>
<tr>
<td></td>
<td>RTC_ConstPeriodInterruptEnable</td>
<td>Sets the cycle of the interrupts INTRTC0, then starts the cyclic interrupt function.</td>
</tr>
<tr>
<td></td>
<td>RTC_ConstPeriodInterruptDisable</td>
<td>Ends the cyclic interrupt function.</td>
</tr>
<tr>
<td></td>
<td>RTC.AlarmEnable</td>
<td>Starts the alarm interrupt function.</td>
</tr>
<tr>
<td></td>
<td>RTC.AlarmDisable</td>
<td>Ends the alarm interrupt function.</td>
</tr>
<tr>
<td></td>
<td>RTC.AlarmSet</td>
<td>Sets the alarm conditions (weekday, hour, minute).</td>
</tr>
<tr>
<td></td>
<td>RTC.AlarmGet</td>
<td>Reads the alarm conditions (weekday, hour, minute).</td>
</tr>
<tr>
<td></td>
<td>RTCTimeIntervalStart</td>
<td>Starts the interval interrupt function.</td>
</tr>
<tr>
<td></td>
<td>RTCTimeIntervalStop</td>
<td>Ends the interval interrupt function.</td>
</tr>
<tr>
<td></td>
<td>RTCTimeIntervalInterruptEnable</td>
<td>Sets the cycle of the interrupts INTRTC2, then starts the interval interrupt function.</td>
</tr>
<tr>
<td></td>
<td>RTCTimeIntervalInterruptDisable</td>
<td>Ends the interval interrupt function.</td>
</tr>
<tr>
<td></td>
<td>RTC_RC1CK1HZ_OutputEnable</td>
<td>Enables output of the real-time counter correction clock (1 Hz) to the RC1CK1HZ pin.</td>
</tr>
<tr>
<td></td>
<td>RTC_RC1CK1HZ_OutputDisable</td>
<td>Disables output of the real-time counter correction clock (1 Hz) to the RC1CK1HZ pin.</td>
</tr>
<tr>
<td></td>
<td>RTC_RC1CKO_OutputEnable</td>
<td>Enables output of the real-time counter clock (32 kHz source) to the RC1CKO pin.</td>
</tr>
<tr>
<td></td>
<td>RTC_RC1CKO_OutputDisable</td>
<td>Disables output of the real-time counter clock (32 kHz source) to the RC1CKO pin.</td>
</tr>
<tr>
<td></td>
<td>RTC_RC1CKDIV_OutputEnable</td>
<td>Enables output of the real-time counter clock (32 kHz cycle) to the RC1CKDIV pin.</td>
</tr>
<tr>
<td></td>
<td>RTC_RC1CKDIV_OutputDisable</td>
<td>Disables output of the real-time counter clock (32 kHz cycle) to the RC1CKDIV pin.</td>
</tr>
<tr>
<td></td>
<td>RTC_RTC1HZ_OutputEnable</td>
<td>Enables output of the real-time counter correction clock (1 Hz) to the RTC1HZ pin.</td>
</tr>
<tr>
<td></td>
<td>RTC_RTC1HZ_OutputDisable</td>
<td>Disables output of the real-time counter correction clock (1 Hz) to the RTC1HZ pin.</td>
</tr>
<tr>
<td></td>
<td>RTC_RTCCL_OutputEnable</td>
<td>Enables output of the real-time counter clock (32 kHz source) to the RTCCCL pin.</td>
</tr>
<tr>
<td></td>
<td>RTC_RTCCL_OutputDisable</td>
<td>Disables output of the real-time counter clock (32 kHz source) to the RTCCCL pin.</td>
</tr>
<tr>
<td></td>
<td>RTC_RTCDIV_OutputEnable</td>
<td>Enables output of the real-time counter clock (32 kHz cycle) to the RTCDIV pin.</td>
</tr>
</tbody>
</table>
### Real-time Clock

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTC_RTCDIV_OutputDisable</td>
<td>Disables output of the real-time counter clock (32 kHz cycle) to the RTCDIV pin.</td>
</tr>
<tr>
<td>RTC_ChangeCorrectionValue</td>
<td>Changes the timing and correction value for correcting clock errors.</td>
</tr>
</tbody>
</table>

### Real-Time Output

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

### DMA

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAn_Init</td>
<td>Performs initialization necessary to control DMA controller functions.</td>
</tr>
<tr>
<td>DMAn_UserInit</td>
<td>Performs user-defined initialization relating to the DMA controller.</td>
</tr>
<tr>
<td>DMAn_Enable</td>
<td>Enables operation of channel n.</td>
</tr>
<tr>
<td>DMAn_Disable</td>
<td>Disables operation of channel n.</td>
</tr>
<tr>
<td>DMAn_CheckStatus</td>
<td>Reads the transfer status (transfer complete/transfer ongoing).</td>
</tr>
<tr>
<td>DMAn_SetData</td>
<td>Sets the RAM address of the transfer source/destination, and the number of times the data has been transferred.</td>
</tr>
<tr>
<td>DMAn_SoftwareTriggerOn</td>
<td>Uses a software trigger as a DMA transfer start trigger.</td>
</tr>
</tbody>
</table>

### LVI

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>LVI_Init</td>
<td>Performs initialization necessary to control low-voltage detector functions.</td>
</tr>
<tr>
<td>LVI_UserInit</td>
<td>Performs user-defined initialization relating to the low-voltage detector.</td>
</tr>
<tr>
<td>LVI_InterruptModeStart</td>
<td>Starts low-voltage detection (when in interrupt generation mode).</td>
</tr>
<tr>
<td>LVI_ResetModeStart</td>
<td>Starts low-voltage detection (when in internal reset mode).</td>
</tr>
<tr>
<td>LVI_Start</td>
<td>Starts low-voltage detection.</td>
</tr>
<tr>
<td>LVI_Stop</td>
<td>Stops low-voltage detection.</td>
</tr>
</tbody>
</table>
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)]**
  
<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- **(6) [Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- **(7) [Example]**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
(5) [Argument(s)]
API function arguments are explained in the following format.

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>(b)</td>
<td>(c)</td>
</tr>
</tbody>
</table>

(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.

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>(b)</td>
</tr>
</tbody>
</table>

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

```c
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]**

```c
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]**

```c
void systeminit ( void ) {
    CG_ReadResetSource (); /* Perform process according to reset signal trigger */
    ......  
}
```

**[CG_system_user.c]**

```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]

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

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>enum ClockMode mode;</td>
<td>CPU clock type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAINOSCCLK: Main clock oscillator (fXX)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUBCLK: Subclock oscillator (fXT)</td>
</tr>
</tbody>
</table>

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ERROR1</td>
<td>Exit with error (abend)</td>
</tr>
<tr>
<td></td>
<td>- Cannot change from the subclock oscillator (fXT) to the main clock oscillator (fXX).</td>
</tr>
<tr>
<td>MD_ERROR2</td>
<td>Exit with error (abend)</td>
</tr>
<tr>
<td></td>
<td>- Cannot change from the main clock oscillator (fXX) to the subclock oscillator (fXT).</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
**CG_ChangeFrequency**

Changes the CPU clock division ratio.

**[Classification]**

CG_system.c

**[Syntax]**

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

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>enum CPUClock clock;</td>
<td>Division ratio type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SYSTEMCLOCK: fxx</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SYSONEHALF: fxx/2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SYSONEFOURTH: fxx/4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SYSONEEIGHTH: fxx/8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SYSONESIXTEENTH: fxx/16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SYSONETHIRTYSECOND: fxx/32</td>
</tr>
</tbody>
</table>

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

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ERROR</td>
<td>Exit with error (abend)</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
**CG_SelectPowerSaveMode**

Configures the CPU's standby function.

**[Classification]**

CG_system.c

**[Syntax]**

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

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>enum PSLevel level;</td>
<td>Standby function type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[E/Sx3-H][ES/Jx3-E][ES/Jx3-H]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PSSTOP: STOP mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PSHALT: HALT mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PSIDLE1: IDLE1 mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PSIDLE2: IDLE2 mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[ES/Jx3][ES/Jx3-L]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PSSTOP: STOP mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PSHALT: HALT mode</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
Selects the oscillation stabilization time for the X1 oscillator. This will become necessary when STOP mode is released.

[Classification]
  CG_system.c

[Syntax]

```c
#include    "CG_macrodriver.h"
#include    "CG_system.h"

MD_STATUS   CG_SelectStabTime ( enum StabTime waittime );
```

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>enum StabTime waittime;</td>
<td>Oscillation stabilization time type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STLEVEL0: 2^10/fx</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STLEVEL1: 2^11/fx</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STLEVEL2: 2^12/fx</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STLEVEL3: 2^13/fx</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STLEVEL4: 2^14/fx</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STLEVEL5: 2^15/fx</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STLEVEL6: 2^16/fx</td>
</tr>
</tbody>
</table>

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
CG_SelecPllMode

Selects the operation mode of the PLL function.

[Classification]
CG_system.c

[Syntax]

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

[Argument(s)]

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

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ERROR</td>
<td>Exit with error (abend) [ES/Jx3] [ES/Jx3-E] [ES/Jx3-H] [ES/Jx3-L]</td>
</tr>
<tr>
<td>MD_ERROR1</td>
<td>Exit with error (abend) [E/Sx3-H]</td>
</tr>
<tr>
<td></td>
<td>- Cannot change the operation mode.</td>
</tr>
<tr>
<td>MD_ERROR2</td>
<td>Exit with error (abend) [E/Sx3-H]</td>
</tr>
<tr>
<td></td>
<td>- Cannot change to the x4.</td>
</tr>
<tr>
<td>MD_ERROR3</td>
<td>Exit with error (abend) [E/Sx3-H]</td>
</tr>
<tr>
<td></td>
<td>- Cannot change to the x8.</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
CG_SelectSSCGMode

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

[Classification]

CG_system.c

[Syntax]

```c
#include "CG_macrodriver.h"
#include "CG_system.h"

MD_STATUS CG_SelectSSCGMode ( enum SSCGMode sscgmode);
```

[I/O Argument Description]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>enum SSCGMode sscgmode;</td>
<td>Operation mode type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SYSSSCGON: SSCG operation enabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SYSSSCGOFF: SSCG operation stopped</td>
</tr>
</tbody>
</table>

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ERROR</td>
<td>Exit with error (abend)</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
WDT2_Restart

Clears the watchdog timer counter and resumes counting.

[Classification]
CG_system.c

[Syntax]

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

[Argument(s)]
None.

[Return value]
None.
CRC_Start

Begins detection of data-block errors.

[Classification]
CG_system.c

[Syntax]

```c
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]

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

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>UCHAR data</td>
<td>Data to set</td>
</tr>
</tbody>
</table>

[Return value]
None.
**CRC_GetResult**

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

**[Classification]**

CG_system.c

**[Syntax]**

```c
#include "CG_macrodriver.h"

void CRC_GetResult ( USHORT *result );
```

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>USHORT *result</td>
<td>Pointer to area in which to store read calculation results</td>
</tr>
</tbody>
</table>

**[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]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUS_Init</td>
<td>Performs initialization necessary to control external bus interface functions (functions to connect an external bus to areas other than onboard ROM, ROM and RAM).</td>
</tr>
<tr>
<td>BUS_UserInit</td>
<td>Performs user-defined initialization relating to the external bus interface.</td>
</tr>
</tbody>
</table>
**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]**

```c
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]**

```c
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>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PORT_Init</td>
<td>Performs initialization necessary to control port functions.</td>
</tr>
<tr>
<td>PORT_UserInit</td>
<td>Performs user-defined initialization relating to the port.</td>
</tr>
<tr>
<td>PORT_ChangePmnInput</td>
<td>Switches the pin's I/O mode from output mode to input mode.</td>
</tr>
<tr>
<td>PORT_ChangePmnOutput</td>
<td>Switches the pin's I/O mode from input mode to output mode.</td>
</tr>
</tbody>
</table>
**PORT_Init**

Performs initialization necessary to control port functions.

**[Classification]**

CG_port.c

**[Syntax]**

```c
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**

```c
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]**

```c
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]**

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

```c
#include "CG_macrodriver.h"
void PORT_ChangePmnOutput ( BOOL initialValue );
```

- [N-ch open drain output: yes]

```c
#include "CG_macrodriver.h"
void PORT_ChangePmnOutput ( BOOL enableNch, BOOL initialValue );
```

Remark  

\( nm \) is the port number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>BOOL enableNch</td>
<td>Output mode type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_TRUE: N-ch open drain output (VDD withstand voltage) mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_FALSE: Normal output mode</td>
</tr>
<tr>
<td>I</td>
<td>BOOL initialValue</td>
<td>Initial output value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_SET: Output HIGH level &quot;1&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_CLEAR: Output LOW level &quot;0&quot;</td>
</tr>
</tbody>
</table>

[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]

```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"

[CubeSuite+ V1.03.00]

```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]

<table>
<thead>
<tr>
<th>API Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTP_Init</td>
<td>Performs initialization necessary to control the external interrupt INTPn functions.</td>
</tr>
<tr>
<td>INTP_UserInit</td>
<td>Performs user-defined initialization relating to the external interrupt INTPn functions.</td>
</tr>
<tr>
<td>KEY_Init</td>
<td>Performs initialization necessary to control the key interrupt INTKR functions.</td>
</tr>
<tr>
<td>KEY_UserInit</td>
<td>Performs user-defined initialization relating to the key interrupt INTKR functions.</td>
</tr>
<tr>
<td>INT_MaskableInterruptEnable</td>
<td>Enables/disables the acceptance of the maskable interrupts.</td>
</tr>
<tr>
<td>INTPn_Disable</td>
<td>Disables the acceptance of the maskable interrupts INTPn (external interrupt requests).</td>
</tr>
<tr>
<td>INTPn_Enable</td>
<td>Enables the acceptance of the maskable interrupts INTPn (external interrupt requests).</td>
</tr>
<tr>
<td>KEY_Disable</td>
<td>Disables the acceptance of the key interrupts INTKR.</td>
</tr>
<tr>
<td>KEY_Enable</td>
<td>Enables the acceptance of the key interrupts INTKR.</td>
</tr>
</tbody>
</table>
INTP_Init

Performs initialization necessary to control the external interrupt INTPn functions.

[Classification]
CG_int.c

[Syntax]

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

[Argument(s)]
None.

[Return value]
None.
**INTP_UserInit**

Performs user-defined initialization relating to the external interrupt INTPn functions.

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

[Classification]  
CG_int_user.c

[Syntax]

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

[Argument(s)]

None.

[Return value]

None.
Performs initialization necessary to control the key interrupt INTKR functions.

[Classification]
CG_int.c

[Syntax]

```c
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]

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

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

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

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>enum MaskableSource name;</td>
<td>Maskable interrupt type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INT_xxx: Maskable interrupt</td>
</tr>
<tr>
<td>I</td>
<td>BOOL enableflag;</td>
<td>Acceptance enabled/disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_TRUE: Acceptance enabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_FALSE: Acceptance disabled</td>
</tr>
</tbody>
</table>

**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]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>

- [ES/Jx3] [ES/Jx3-L]
None.

[Example 1]

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

```c
#include    "CG_macrodriver.h"
#include    "CG_int.h"
MD_STATUS   INT_MaskableInterruptEnable ( INT_xxx name, MD_FALSE enableflag );
```
Below is an example of enabling acceptance of the maskable interrupt INTP0.

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

```c
#include    "CG_macrodriver.h"
#include    "CG_int.h"
void main ( void ) {
    ......
    INTP_MaskableInterruptEnable ( INT_INTP0, MD_TRUE );   /* Enable acceptance of maskable interrupt INTP0 */
    ......
}
```
INTP\textsubscript{n} Disable

Disables the acceptance of the maskable interrupts INTP\textsubscript{n} (external interrupt requests).

[Classification]
CG\_int.c

[Syntax]

\begin{verbatim}
void INTP\textsubscript{n} Disable ( void );
\end{verbatim}

Remark \textit{n} is the interrupt factor number.

[Argument(s)]
None.

[Return value]
None.
**INTP\textsubscript{n} Enable**

Enables the acceptance of the maskable interrupts INTP\textsubscript{n} (external interrupt requests).

[Classification]

CG\_int\_c

[Syntax]

```c
void INTP\textsubscript{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]

```c
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]

```c
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>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>UARTAn_Init</td>
<td>Performs initialization necessary to control the asynchronous serial interface A (UARTA) functions.</td>
</tr>
<tr>
<td>UARTAn_UserInit</td>
<td>Performs user-defined initialization relating to the asynchronous serial interface A (UARTA).</td>
</tr>
<tr>
<td>UARTAn_Start</td>
<td>Enables asynchronous serial interface A (UARTA).</td>
</tr>
<tr>
<td>UARTAn_Stop</td>
<td>Disables asynchronous serial interface A (UARTA).</td>
</tr>
<tr>
<td>UARTAn_SendData</td>
<td>Starts UARTAn data transmission.</td>
</tr>
<tr>
<td>UARTAn_ReceiveData</td>
<td>Starts UARTAn data reception.</td>
</tr>
<tr>
<td>UARTAn_SendEndCallback</td>
<td>Performs processing in response to the UARTAn consecutive transmission enable interrupt INTUA_nT.</td>
</tr>
<tr>
<td>UARTAn_ReceiveEndCallback</td>
<td>Performs processing in response to the UARTAn reception completion interrupt INTU_AnR.</td>
</tr>
<tr>
<td>UARTAn_ErrorCallback</td>
<td>Performs processing in response to the UARTAn reception error interrupt INTUA_nR (overrun error, framing error, parity error).</td>
</tr>
<tr>
<td>UARTAn_SoftOverRunCallback</td>
<td>Performs processing in response to detection of overrun error.</td>
</tr>
<tr>
<td>UARTBn_Init</td>
<td>Performs initialization necessary to control the asynchronous serial interface B (UARTB) functions.</td>
</tr>
<tr>
<td>UARTBn_UserInit</td>
<td>Performs user-defined initialization relating to the asynchronous serial interface B (UARTB).</td>
</tr>
<tr>
<td>UARTBn_Start</td>
<td>Enables asynchronous serial interface B (UARTB).</td>
</tr>
<tr>
<td>UARTBn_Stop</td>
<td>Disables asynchronous serial interface B (UARTB).</td>
</tr>
<tr>
<td>UARTBn_SendData</td>
<td>Starts UARTBn data transmission.</td>
</tr>
<tr>
<td>UARTBn_ReceiveData</td>
<td>Starts UARTBn data reception.</td>
</tr>
<tr>
<td>UARTBn_SendEndCallback</td>
<td>Performs processing consequent to the transmission enable interrupt INTUBnTIT and the FIFO transmission completion interrupt INTUBnTIF.</td>
</tr>
<tr>
<td>UARTBn_ReceiveEndCallback</td>
<td>Performs processing in response to the reception completion interrupt INTUBnTIR.</td>
</tr>
<tr>
<td>UARTBn_SingleErrorCallback</td>
<td>Performs processing in response to the reception error interrupt INTUBnTIRE (overrun error, framing error, parity error).</td>
</tr>
<tr>
<td>UARTBn_FIFOErrorCallback</td>
<td>Performs processing in response to the reception error interrupt INTUBnTIRE (overrun error, framing error, parity error).</td>
</tr>
<tr>
<td>UARTBn_TimeoutErrorCallback</td>
<td>Performs processing in response to the reception timeout error interrupt INTUBnTITO.</td>
</tr>
<tr>
<td>UARTBn_SoftOverRunCallback</td>
<td>Performs processing in response to detection of overrun error.</td>
</tr>
<tr>
<td>UARTCn_Init</td>
<td>Performs initialization necessary to control the asynchronous serial interface C (UARTC) functions.</td>
</tr>
<tr>
<td>UARTCn_UserInit</td>
<td>Performs user-defined initialization relating to the asynchronous serial interface C (UARTC).</td>
</tr>
<tr>
<td>UARTCn_Start</td>
<td>Enables asynchronous serial interface C (UARTC).</td>
</tr>
<tr>
<td>UARTCn_Stop</td>
<td>Disables asynchronous serial interface C (UARTC).</td>
</tr>
<tr>
<td>API Function Name</td>
<td>Function</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>UARTCn_SendData</td>
<td>Starts UARTCn data transmission.</td>
</tr>
<tr>
<td>UARTCn_ReceiveData</td>
<td>Starts UARTCn data reception.</td>
</tr>
<tr>
<td>UARTCn_SendEndCallback</td>
<td>Performs processing in response to the UARTCn consecutive transmission enable interrupt INTUCnT.</td>
</tr>
<tr>
<td>UARTCn_ReceiveEndCallback</td>
<td>Performs processing in response to the UARTCn reception completion interrupt INTUCnR.</td>
</tr>
<tr>
<td>UARTCn_ErrorCallback</td>
<td>Performs processing in response to the UARTCn reception error interrupt INTUCnR (overrun error, framing error, parity error).</td>
</tr>
<tr>
<td>UARTCn_SoftOverRunCallback</td>
<td>Performs processing in response to detection of overrun error.</td>
</tr>
<tr>
<td>CSIBn_Init</td>
<td>Performs initialization necessary to control the 3-wire variable-length serial I/O B (CSIB) functions.</td>
</tr>
<tr>
<td>CSIBn_UserInit</td>
<td>Performs user-defined initialization relating to the 3-wire variable-length serial I/O B (CSIB).</td>
</tr>
<tr>
<td>CSIBn_Start</td>
<td>Enables 3-wire variable-length serial I/O B (CSIB).</td>
</tr>
<tr>
<td>CSIBn_Stop</td>
<td>Disables 3-wire variable-length serial I/O B (CSIB).</td>
</tr>
<tr>
<td>CSIBn_SendData</td>
<td>Starts CSIB data transmission.</td>
</tr>
<tr>
<td>CSIBn_ReceiveData</td>
<td>Starts CSIB data reception.</td>
</tr>
<tr>
<td>CSIBn_SendReceiveData</td>
<td>Starts CSIB data transmission/reception.</td>
</tr>
<tr>
<td>CSIBn_SendEndCallback</td>
<td>Performs processing in response to the CSIBn reception completion interrupt INTCBnR or the CSIBn consecutive transmission write enable interrupt INTCBnT.</td>
</tr>
<tr>
<td>CSIBn_ReceiveEndCallback</td>
<td>Performs processing in response to the CSIBn reception completion interrupt INTCBnR.</td>
</tr>
<tr>
<td>CSIBn_ErrorCallback</td>
<td>Performs processing in response to the CSIBn reception error interrupt INTCBnR (overrun error).</td>
</tr>
<tr>
<td>CSIEn_Init</td>
<td>Performs initialization necessary to control the 3-wire variable-length serial I/O E (CSIIE) functions.</td>
</tr>
<tr>
<td>CSIEn_UserInit</td>
<td>Performs user-defined initialization relating to the 3-wire variable-length serial I/O E (CSIIE).</td>
</tr>
<tr>
<td>CSIEn_Start</td>
<td>Enables 3-wire variable-length serial I/O E (CSIIE).</td>
</tr>
<tr>
<td>CSIEn_Stop</td>
<td>Disables 3-wire variable-length serial I/O E (CSIIE).</td>
</tr>
<tr>
<td>CSIEn_SendData</td>
<td>Starts CSIE data transmission.</td>
</tr>
<tr>
<td>CSIEn_ReceiveData</td>
<td>Starts CSIE data reception.</td>
</tr>
<tr>
<td>CSIEn_SendReceiveData</td>
<td>Starts CSIE data transmission/reception.</td>
</tr>
<tr>
<td>CSIEn_SendEndCallback</td>
<td>Performs processing in response to the CSIEn transmission/reception completion interrupt INTCEnT.</td>
</tr>
<tr>
<td>CSIEn_ReceiveEndCallback</td>
<td>Performs processing in response to the CSIEn transmission/reception completion interrupt INTCEnT.</td>
</tr>
<tr>
<td>CSIEn_ErrorCallback</td>
<td>Performs processing in response to the CSIEnBUF overflow interrupt INTCEnTIOF.</td>
</tr>
<tr>
<td>CSIFn_Init</td>
<td>Performs initialization necessary to control the 3-wire variable-length serial I/O F (CSIIF) functions.</td>
</tr>
<tr>
<td>CSIFn_UserInit</td>
<td>Performs user-defined initialization relating to the 3-wire variable-length serial I/O F (CSIIF).</td>
</tr>
<tr>
<td>API Function Name</td>
<td>Function</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CSIFn_Start</td>
<td>Enables 3-wire variable-length serial I/O F (CSIF).</td>
</tr>
<tr>
<td>CSIFn_Stop</td>
<td>Disables 3-wire variable-length serial I/O F (CSIF).</td>
</tr>
<tr>
<td>CSIFn_SendData</td>
<td>Starts CSIF data transmission.</td>
</tr>
<tr>
<td>CSIFn_ReceiveData</td>
<td>Starts CSIF data reception.</td>
</tr>
<tr>
<td>CSIFn_SendReceiveData</td>
<td>Starts CSIF data transmission/reception.</td>
</tr>
<tr>
<td>CSIFn_SendEndCallback</td>
<td>Performs processing in response to the CSIFn transmission/reception completion interrupt INTCFnT.</td>
</tr>
<tr>
<td>CSIFn_ReceiveEndCallback</td>
<td>Performs processing in response to the CSIFn transmission/reception completion interrupt INTCFnT.</td>
</tr>
<tr>
<td>CSIFn_ErrorCallback</td>
<td>Performs processing in response to the CSIFn reception error interrupt INTCFnR (overrun error).</td>
</tr>
<tr>
<td>IIC0n_Init</td>
<td>Performs initialization necessary to control the IIC bus functions.</td>
</tr>
<tr>
<td>IIC0n_UserInit</td>
<td>Performs user-defined initialization relating to the IIC bus.</td>
</tr>
<tr>
<td>IIC0n_Stop</td>
<td>Ends IIC0n communication.</td>
</tr>
<tr>
<td>IIC0n_StopCondition</td>
<td>Generates a stop condition.</td>
</tr>
<tr>
<td>IIC0n_MasterSendStart</td>
<td>Starts IIC0n master transmission.</td>
</tr>
<tr>
<td>IIC0n_MasterReceiveStart</td>
<td>Starts IIC0n master reception.</td>
</tr>
<tr>
<td>IIC0n_MasterSendEndCallback</td>
<td>Performs processing in response to the IIC0n master transfer completion interrupt INTIICn.</td>
</tr>
<tr>
<td>IIC0n_MasterReceiveEndCallback</td>
<td>Performs processing in response to the IIC0n master transfer completion interrupt INTIICn.</td>
</tr>
<tr>
<td>IIC0n_MasterErrorCallback</td>
<td>Performs processing in response to detection of error in IIC0n master communication.</td>
</tr>
<tr>
<td>IIC0n_SlaveSendStart</td>
<td>Starts IIC0n slave transmission.</td>
</tr>
<tr>
<td>IIC0n_SlaveReceiveStart</td>
<td>Starts IIC0n slave reception.</td>
</tr>
<tr>
<td>IIC0n_SlaveSendEndCallback</td>
<td>Performs processing in response to the IIC0n slave transfer completion interrupt INTI-ICn.</td>
</tr>
<tr>
<td>IIC0n_SlaveReceiveEndCallback</td>
<td>Performs processing in response to the IIC0n slave transfer completion interrupt INTI-ICn.</td>
</tr>
<tr>
<td>IIC0n_SlaveErrorCallback</td>
<td>Performs processing in response to detection of error in IIC0n slave communication.</td>
</tr>
<tr>
<td>IIC0n_GetStopConditionCallback</td>
<td>Performs processing in response to detection of stop condition.</td>
</tr>
</tbody>
</table>
**UARTA\textsubscript{n} Init**

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

**Classification**

CG\_serial.c

**Syntax**

```c
void UARTA\textsubscript{n} 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]**

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

**Remark**  \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
Enables asynchronous serial interface A (UARTA).

[Classification]
CG_serial.c

[Syntax]

```c
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]**

```c
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**

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

**Remark**  
$n$ is the channel number.

**Argument(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>UCHAR *txbuf;</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>USHORT txnum;</td>
<td>Total amount of data to send</td>
</tr>
</tbody>
</table>

**Return value**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
<tr>
<td>MD_DATAEXISTS</td>
<td>Exit with error (abend)</td>
</tr>
<tr>
<td></td>
<td>- Hold a next transmission data in the UARTAn register.</td>
</tr>
</tbody>
</table>
**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]**

```c
#include "CG_macrodriver.h"
MD_STATUS UARTAn.ReceiveData ( UCHAR *rbuf, USHORT rxnum );
```

**Remark**

`n` is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>UCHAR rbuf;</td>
<td>Pointer to a buffer to store the received data</td>
</tr>
<tr>
<td>I</td>
<td>USHORT rxnum;</td>
<td>Total amount of data to receive</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
**UARTAn_SendEndCallback**

Performs processing in response to the UARTAn consecutive transmission enable interrupt INTUA\textsubscript{n}T.

**Remark**  This API function is called as the callback routine of interrupt process MD\_INTUA\textsubscript{n}T corresponding to the UARTAn consecutive transmission enable interrupt INTUA\textsubscript{n}T (performed when total number of UARTAn transmissions specified by **UARTAn\_SendData** parameter txnum has been completed).

**[Classification]**

CG\_serial\_user.c

**[Syntax]**

```c
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 INTUA\(_n\)R.

**Remark** This API function is called as the callback routine of interrupt process MD_INTUA\(_n\)R corresponding to the UARTAn reception completion interrupt INTUA\(_n\)R (performed when total number of UARTAn receptions specified by UARTAn_ReceiveData parameter \(rxnum\) has been completed).

**[Classification]**
CG_serial_user.c

**[Syntax]**

```c
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]**

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

**Remark** $n$ is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>UCHAR err_type;</td>
<td>Trigger for UARTAn reception error interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>000000xx1B: Overrun error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00000x1xB: Framing error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>000001xxB: Parity error</td>
</tr>
</tbody>
</table>

**[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_INTUARTnR corresponding to the UARTAn reception error interrupt INTUARTnR (process performed when the amount of data received is greater than the parameter rxnum specified for UARTAn_ReceiveData).

**[Classification]**
CG_serial_user.c

**[Syntax]**

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

**Remark** `n` is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>UCHAR rx_data;</td>
<td>Received data</td>
</tr>
</tbody>
</table>

**[Return value]**
None.
**UARTBn_Init**

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

**[Classification]**

CG_serial.c

**[Syntax]**

```c
void UARTBn_Init ( void );
```

**Remark**  
$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**UARTB\textsubscript{n} UserInit**

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

**Remark**  This API function is called as the UARTB\textsubscript{n} Init callback routine.

**[Classification]**

CG_serial_user.c

**[Syntax]**

```c
void UARTB\textsubscript{n} UserInit ( void );
```

**Remark**  \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**UARTB\textsubscript{n} \_Start**

Enables asynchronous serial interface B (UARTB).

**[Classification]**
CG\_serial.c

**[Syntax]**

```c
void UARTB\textsubscript{n} \_Start ( void );
```

**Remark**  \( n \) is the channel number.

**[Argument(s)]**
None.

**[Return value]**
None.
Disables asynchronous serial interface B (UARTB).

**[Classification]**

CG_serial.c

**[Syntax]**

```c
void UARTBn_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)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>UCHAR *txbuf;</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>USHORT txnum;</td>
<td>Total amount of data to send</td>
</tr>
</tbody>
</table>

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
<tr>
<td></td>
<td>- <code>txnum</code> is not a multiple of transmit FIFO as trigger.</td>
</tr>
<tr>
<td>MD_DATAEXISTS</td>
<td>Exit with error (abend)</td>
</tr>
<tr>
<td></td>
<td>- Hold a next transmission data in the UBNTX register.</td>
</tr>
</tbody>
</table>
**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]**

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

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>UCHAR *rxbuf;</td>
<td>Pointer to a buffer to store the received data</td>
</tr>
<tr>
<td>I</td>
<td>USHORT rxnum;</td>
<td>Total amount of data to receive</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
<tr>
<td></td>
<td>- <code>rxnum</code> is not a multiple of receive FIFO as trigger.</td>
</tr>
</tbody>
</table>
UARTB\textsubscript{n} \_SendEndCallback

Performs processing consequent to the transmission enable interrupt INTUB\textsubscript{n}TIT and the FIFO transmission completion interrupt INTUB\textsubscript{n}TIF.

**Remark** This API function is called as a callback routine of the interrupt process MD\_INTUB\textsubscript{n}TIT corresponding to a transmission enable interrupt INTUB\textsubscript{n}TIT of UART\textsubscript{b}(single mode), and interrupt process MD\_INTUB\textsubscript{n}TIF corresponding to a FIFO transmission completion interrupt INTUB\textsubscript{n}TIF of UART\textsubscript{b}(FIFO mode) (performed when total number of UART\textsubscript{b} transmissions specified by UARTB\textsubscript{n}\_SendData parameter \textit{txnum} has been completed).

**[Classification]**
CG\_serial\_user.c

**[Syntax]**

```c
void UARTB\textsubscript{n} \_SendEndCallback ( void );
```

**Remark** \( n \) is the channel number.

**[Argument(s)]**
None.

**[Return value]**
None.
**UARTB_n_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]**

```c
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]**

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

**Remark**  \( n \) is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>UCHAR err_type;</td>
<td>Trigger for reception error interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00000xx1B: Overrun error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>000001xB: Framing error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>000001xxB: Parity error</td>
</tr>
</tbody>
</table>

**[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**

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

**Remark** \( n \) is the channel number.

**Argument(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>UCHAR err_type1;</td>
<td>Trigger for reception error interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00001000B: Overrun error</td>
</tr>
<tr>
<td>O</td>
<td>UCHAR err_type2;</td>
<td>Trigger for reception error interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0000000x1B: Framing error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00000001xB: Parity error</td>
</tr>
</tbody>
</table>

**Return value**

None.
UARTB\text{n}_TimeoutErrorCallback

Performs processing in response to the reception timeout error interrupt INTUB\text{n}TITO.

**Remark**  This API function is called as a callback routine of the interrupt process MD_INTUB\text{n}TITO corresponding to a reception timeout interrupt INTUB\text{n}TITO of UARTB\text{n} (FIFO mode).

[Classification]

CG\_serial\_user.c

[Syntax]

```c
void UARTB\text{n}\_TimeoutErrorCallback ( void );
```

**Remark**  \( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
**UARTB\textsubscript{n} _SoftOverRunCallback**

Performs processing in response to detection of overrun error.

**Remark** This API function is called as the callback routine of interrupt process MD\_INTUB\textsubscript{n}TIRE corresponding to the reception error interrupt INTUB\textsubscript{n}TIRE (process performed when the amount of data received is greater than the parameter \textit{rxnum} specified for UARTB\textsubscript{n}\_ReceiveData).

**[Classification]**
CG\_serial\_user.c

**[Syntax]**

```c
#include "CG_macerdriver.h"
void UARTB\textsubscript{n} _SoftOverRunCallback ( UCHAR rx_data );
```

**Remark** \( n \) is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>UCHAR \textit{rx_data};</td>
<td>Received data</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
**UARTCn_Init**

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

[**Classification**]

CG_serial.c

[**Syntax**]

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

**Remark**  
$n$ is the channel number.

[**Argument(s)**]

None.

[**Return value**]

None.
<table>
<thead>
<tr>
<th>UARTCn_UserInit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performs user-defined initialization relating to the asynchronous serial interface C (UARTC).</td>
</tr>
</tbody>
</table>

**Remark**  This API function is called as the UARTCn_Init callback routine.

**[Classification]**

CG_serial_user.c

**[Syntax]**

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

**Remark**  

`n` is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
UARTC\textsubscript{n} \_Start

Enables asynchronous serial interface C (UARTC).

[Classification]
CG\_serial.c

[Syntax]

```c
void UARTC\textsubscript{n} \_Start ( void );
```

Remark \textit{n} is the channel number.

[Argument(s)]
None.

[Return value]
None.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Classification</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>UARTCn_Stop</td>
<td>Disables asynchronous serial interface C (UARTC).</td>
<td></td>
<td><code>void UARTCn_Stop ( void );</code></td>
</tr>
</tbody>
</table>

**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]

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

Remark  n is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>UCHAR *txbuf;</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>USHORT txnum;</td>
<td>Total amount of data to send</td>
</tr>
</tbody>
</table>

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
<tr>
<td>MD_DATAEXISTS</td>
<td>Exit with error (abend)</td>
</tr>
<tr>
<td></td>
<td>- Hold a next transmission data in the UCTX register.</td>
</tr>
</tbody>
</table>
UARTC\textsubscript{n} \_ReceiveData

Starts UART\textsubscript{Cn} data reception.

**Remarks**

1. This API function performs byte-level UART\textsubscript{Cn} reception the number of times specified by the parameter \textit{rxnum}, and stores the data in the buffer specified by the parameter \textit{rxbuf}.

2. Actual UART\textsubscript{Cn} reception starts after this API function is called, and \textit{UARTC\textsubscript{n} \_Start} is then called.

[Classification]

CG\_serial.c

[Syntax]

```c
#include "CG\_macrodriver.h"
MD\_STATUS UARTC\textsubscript{n} \_ReceiveData ( UCHAR *rxbuf, USHORT rxnum );
```

**Remark** \( n \) is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>UCHAR *rxbuf;</td>
<td>Pointer to a buffer to store the received data</td>
</tr>
<tr>
<td>I</td>
<td>USHORT rxnum;</td>
<td>Total amount of data to receive</td>
</tr>
</tbody>
</table>

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
**UARTC\(n\) SendEndCallback**

Performs processing in response to the UARTC\(n\) consecutive transmission enable interrupt INTUC\(n\)T.

**Remark**
This API function is called as the callback routine of interrupt process MD_INTUC\(n\)T corresponding to the UARTC\(n\) consecutive transmission enable interrupt INTUC\(n\)T (performed when total number of UARTC\(n\) transmissions specified by UARTC\(n\)_SendData parameter \(txnum\) has been completed).

### [Classification]

CG_serial_user.c

### [Syntax]

```c
void UARTC\(n\)_SendEndCallback ( void );
```

**Remark**
\(n\) is the channel number.

### [Argument(s)]

None.

### [Return value]

None.
**UARTC\text{n}_ReceiveEndCallback**

Performs processing in response to the UARTC\text{n} reception completion interrupt INTUC\text{n}R.

**Remark**  
This API function is called as the callback routine of interrupt process MD_INTUC\text{n}R corresponding to the UARTC\text{n} reception completion interrupt INTUC\text{n}R (performed when total number of UARTC\text{n} receptions specified by UARTC\text{n}_ReceiveData parameter rxnum has been completed).

**Classification**

CG_serial_user.c

**Syntax**

```c
void UARTC\text{n}_ReceiveEndCallback ( void );
```

**Remark**  
\( n \) is the channel number.

**Argument(s)**

None.

**Return value**

None.
UARTC\textsubscript{n} ErrorCallback

Performs processing in response to the UARTC\textsubscript{n} reception error interrupt INTUC\textsubscript{R} (overrun error, framing error, parity error).

**Remark** This API function is called as the callback routine of interrupt process MD\_INTUC\textsubscript{R} corresponding to the UARTC\textsubscript{n} reception error interrupt INTUC\textsubscript{R}.

**Classification**
CG\_serial\_user.c

**Syntax**

```c
#include    "CG_macrodriver.h"
void    UARTC\textsubscript{n} ErrorCallback ( UCHAR err_type );
```

**Remark** \( n \) is the channel number.

**Argument(s)**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>UCHAR err_type;</td>
<td>Trigger for UARTC\textsubscript{n} reception error interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00000\textsubscript{XX}1B: Overrun error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00000\textsubscript{XX}1xB: Framing error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>000001\textsubscript{XX}B: Parity error</td>
</tr>
</tbody>
</table>

**Return value**

None.
**UARTC\textsubscript{n} _SoftOverRunCallback**

Performs processing in response to detection of overrun error.

**Remark**  This API function is called as the callback routine of interrupt process MD\_INTUC\textsubscript{n}R corresponding to the UARTC\textsubscript{n} reception error interrupt INTUC\textsubscript{n}R (process performed when the amount of data received is greater than the parameter \textit{rxnum} specified for UARTC\_ReceiveData).

**[Classification]**

CG\_serial\_user.c

**[Syntax]**

```c
#include    "CG_macrodriver.h"
void    UARTC\textsubscript{n} _SoftOverRunCallback ( UCHAR rx_data );
```

**Remark**  \( n \) is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>UCHAR \textit{rx_data};</td>
<td>Received data</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Classification</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSIBn_Init</td>
<td>Performs initialization necessary to control the 3-wire variable-length serial I/O B (CSIB) functions.</td>
<td>CG_serial.c</td>
<td></td>
</tr>
</tbody>
</table>

**[Argument(s)]**

None.

**[Return value]**

None.
### CSIB\textsubscript{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\textsubscript{n} Init callback routine.

**[Classification]**

CG\_serial\_user.c

**[Syntax]**

```c
void CSIB\textsubscript{n} UserInit ( void );
```

**Remark** \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
CSIBn_Start

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

[Classification]
CG_serial.c

[Syntax]

```c
void CSIBn_Start ( void );
```

Remark  

*n* is the channel number.

[Argument(s)]

None.

[Return value]

None.
Disables 3-wire variable-length serial I/O B (CSIB).

[Classification]
CG_serial.c

[Syntax]

```c
void CSIBn_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]**

```c
#include "CG_macrodriver.h"

MD_STATUS CSIBn_SendData ( UCHAR *txbuf, USHORT txnum );
```

**Remark**  $n$ is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>UCHAR *txbuf;</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>USHORT txnum;</td>
<td>Total amount of data to send</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
CSIBn_ReceiveData

Starts CSIBn data reception.

**Remarks**
1. This API function performs byte-level CSIBn 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 CSIBn reception, `CSIBn_Start` must be called before this API function is called.

**[Classification]**
CG_serial.c

**[Syntax]**

```c
#include "CG_macrodriver.h"

MD_STATUS CSIBn_ReceiveData ( UCHAR *rxbuf, USHORT rxnum );
```

**[Remark]**
`n` is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>UCHAR *rxbuf;</td>
<td>Pointer to a buffer to store the received data</td>
</tr>
<tr>
<td>I</td>
<td>USHORT rxnum;</td>
<td>Total amount of data to receive</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
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]

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

Remark   \( n \) is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>UCHAR *txbuf; Pointer to a buffer to store the received data</td>
</tr>
<tr>
<td>I</td>
<td>USHORT txnum; Total amount of data to receive</td>
</tr>
<tr>
<td>I</td>
<td>UCHAR *rxbuf; Pointer to a buffer storing the transmission data</td>
</tr>
</tbody>
</table>

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>

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

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

**[Classification]**

CG\_serial\_user.c

**[Syntax]**

```c
void CSIB\textit{n}\_SendEndCallback ( void );
```

**Remark**  \textit{n} is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**CSIBn_ReceiveEndCallback**

Performs processing in response to the CSIBn reception completion interrupt INTCBnR.

**Remark** This API function is called as the callback routine (process performed when the total number of CSIBn receptions specified in the parameter *rxnum* for `CSIBn_ReceiveData` or `CSIBn_SendReceiveData` has been completed) of interrupt process MD_INTCBnR corresponding to the CSIBn reception completion interrupt INTCBnR.

**[Classification]**

CG_serial_user.c

**[Syntax]**

```c
void CSIBn_ReceiveEndCallback ( void );
```

**Remark**  *n* is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**CSIB\textsubscript{n} ErrorCallback**

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

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

**[Classification]**

CG\_serial\_user.c

**[Syntax]**

```c
void CSIB\textsubscript{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]**

```c
void CSIE\(_n\)_Init ( void );
```

**Remark**  \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**CSIEn_UserInit**

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

**Remark**  This API function is called as the CSIEn_Init callback routine.

**[Classification]**

CG_serial_user.c

**[Syntax]**

```c
void CSIEn_UserInit ( void );
```

**Remark**  n is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
CSIE\textsubscript{n} _Start

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

[Classification]
CG\_serial.c

[Syntax]

\begin{verbatim}
void CSIE\textsubscript{n} _Start ( void );
\end{verbatim}

Remark \( n \) is the channel number.

[Argument(s)]
None.

[Return value]
None.
**Disables 3-wire variable-length serial I/O E (CSIE).**

**[Classification]**
- CG_serial.c

**[Syntax]**

```c
void CSIE\_n\_Stop ( void );
```

**Remark**

\( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
CSIE\textsubscript{n} \_SendData

Starts CSIE\textsubscript{n} data transmission.

**Remarks 1.** This API function repeats the byte-level CSIE\textsubscript{n} transmission from the buffer specified in parameter \texttt{txbuf} the number of times specified in parameter \texttt{txnum}.

2. When performing a CSIE\textsubscript{n} transmission, \texttt{CSIEn\_Start} must be called before this API function is called.

**[Classification]**  
CG\_serial.c

**[Syntax]**

```c
#include "CG_macrodriver.h"

MD\_STATUS CSIE\textsubscript{n} \_SendData ( UCHAR *txbuf, USHORT txnum );
```

**Remark**  \texttt{n} is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>UCHAR *txbuf;</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>USHORT txnum;</td>
<td>Total amount of data to send</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
**CSIE_n.ReceiveData**

Starts CSIE\textsubscript{n} data reception.

**Remarks**

1. This API function performs byte-level CSIE\textsubscript{n} reception the number of times specified by the parameter \textit{rxnum}, and stores the data in the buffer specified by the parameter \textit{rxbuf}.
2. When performing a CSIE\textsubscript{n} reception, \texttt{CSIE_n.Start} must be called before this API function is called.

**[Classification]**

CG\_serial\_c

**[Syntax]**

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

**Remark**  \( n \) is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>UCHAR *rxbuf;</td>
<td>Pointer to a buffer to store the received data</td>
</tr>
<tr>
<td>I</td>
<td>USHORT rxnum;</td>
<td>Total amount of data to receive</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
Starts CSIE\textsubscript{n} data transmission/reception.

**Remarks**

1. This API function repeats the byte-level CSIE\textsubscript{n} transmission from the buffer specified in parameter \textit{txbuf} the number of times specified in parameter \textit{txnum}.
2. This API function performs byte-level CSIE\textsubscript{n} reception the number of times specified by the parameter \textit{txnum}, and stores the data in the buffer specified by the parameter \textit{rxbuf}.
3. When performing a CSIE\textsubscript{n} transmission/reception, \texttt{CSIEn_Start} must be called before this API function is called.

**[Classification]**

CG\_serial.c

**[Syntax]**

```c
#include "CG_macrodriver.h"
MD_STATUS CSIE\textsubscript{n}_SendReceiveData ( UCHAR *txbuf, USHORT txnum, UCHAR *rxbuf );
```

**Remark** \( n \) is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>UCHAR *txbuf;</td>
<td>Pointer to a buffer to store the received data</td>
</tr>
<tr>
<td>I</td>
<td>USHORT txnum;</td>
<td>Total amount of data to receive</td>
</tr>
<tr>
<td>I</td>
<td>UCHAR *rxbuf;</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
**CSIE\textsubscript{n} SendEndCallback**

Performs processing in response to the CSIE\textsubscript{n} transmission/reception completion interrupt INTCE\textsubscript{n}T.

**Remark** This API function is called as the callback routine (process performed when the total number of CSIE\textsubscript{n} transmissions specified in the parameter \textit{txnum} for CSIE\textsubscript{n}SendData or CSIE\textsubscript{n}SendReceiveData has been completed) of interrupt process MD_INTCE\textsubscript{n}T corresponding to the CSIE\textsubscript{n} transmission/reception completion interrupt INTCE\textsubscript{n}T.

[Classification]
CG\_serial\_user.c

[Syntax]

```c
void CSIE\textsubscript{n} SendEndCallback ( void );
```

**Remark** \textit{n} is the channel number.

[Argument(s)]
None.

[Return value]
None.
**CSIE\textsubscript{n} ReceiveEndCallback**

Performs processing in response to the CSIE\textsubscript{n} transmission/reception completion interrupt INTCE\textsubscript{n}T.

**Remark**  This API function is called as the callback routine (process performed when the total number of CSIE\textsubscript{n} receptions specified in the parameter \textit{rxnum} for CSIE\textsubscript{n} ReceiveData or CSIE\textsubscript{n} SendReceiveData has been completed) of interrupt process MD_INTCE\textsubscript{n}T corresponding to the CSIE\textsubscript{n} transmission/reception completion interrupt INTCE\textsubscript{n}T.

**[Classification]**

CD_serial_user.c

**[Syntax]**

```c
void CSIE\textsubscript{n} ReceiveEndCallback ( void );
```

**Remark**  \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
Performs processing in response to the CSIE\textsubscript{n}BUF overflow interrupt INTCE\textsubscript{n}TIOF.

**Remark** This API function is called as the callback routine of interrupt process MD\_INTCE\textsubscript{n}TIOF corresponding to the CSIE\textsubscript{n}BUF overflow interrupt INTCE\textsubscript{n}TIOF.

**[Classification]**
CG\_serial\_user.c

**[Syntax]**

```c
void CSIE\textsubscript{n} ErrorCallback ( void );
```

**Remark** \( n \) is the channel number.

**[Argument(s)]**
None.

**[Return value]**
None.
**CSIF\textsubscript{n} \_Init**

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

[**Classification**]

CG\_serial.c

[**Syntax**]

```c
void    CSIF\textsubscript{n} \_Init ( void );
```

**Remark** \textit{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.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSIFn_Start</td>
<td>Enables 3-wire variable-length serial I/O F (CSIF).</td>
</tr>
</tbody>
</table>

**[Classification]**

CG_serial.c

**[Syntax]**

```c
void CSIFn_Start ( void );
```

**Remark**  

$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
Disables 3-wire variable-length serial I/O F (CSIF).

[Classification]
CG_serial.c

[Syntax]

```c
void CSIFn_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]**

```c
#include "CG_macrodriver.h"

MD_STATUS CSIFn_SendData ( UCHAR *txbuf, USHORT txnum );
```

**Remark**  
'n' is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>UCHAR *txbuf;</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>USHORT txnum;</td>
<td>Total amount of data to send</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
**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]**

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

**Remark**  
`n` is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>UCHAR *rxbuf;</td>
<td>Pointer to a buffer to store the received data</td>
</tr>
<tr>
<td>I</td>
<td>USHORT rxnum;</td>
<td>Total amount of data to receive</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
**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]**

```c
#include "CG_macrodriver.h"

MD_STATUS CSIFn_SendReceiveData ( UCHAR *txbuf, USHORT txnum, UCHAR *rxbuf );
```

**Remark**  
$n$ is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>UCHAR *txbuf;</td>
<td>Pointer to a buffer to store the received data</td>
</tr>
<tr>
<td>I</td>
<td>USHORT txnum;</td>
<td>Total amount of data to receive</td>
</tr>
<tr>
<td>I</td>
<td>UCHAR *rxbuf;</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
**CSIFn_SendEndCallback**

Performs processing in response to the CSIFn transmission/reception completion interrupt INTCF\textsuperscript{n}T.

**Remark**  This API function is called as the callback routine (process performed when the total number of CSIFn transmissions specified in the parameter \textit{txnum} for \texttt{CSIFn_SendData} or \texttt{CSIFn_SendReceiveData} has been completed) of interrupt process MD_INTCF\textsuperscript{n}T corresponding to the CSIFn transmission/reception completion interrupt INTCF\textsuperscript{n}T.

**[Classification]**

CG\_serial\_user.c

**[Syntax]**

```c
void CSIFn_SendEndCallback ( void );
```

**Remark**  \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**CSIFn_ReceiveEndCallback**

Performs processing in response to the CSIFn transmission/reception completion interrupt INTCFnT.

**Remark** This API function is called as the callback routine (process performed when the total number of CSIFn receptions specified in the parameter \( rxnum \) for CSIFn_ReceiveData or CSIFn_SendReceiveData has been completed) of interrupt process MD_INTCFnT corresponding to the CSIFn transmission/reception completion interrupt INTCFnT.

**[Classification]**  
CG_serial_user.c

**[Syntax]**

```c
void    CSIFn_ReceiveEndCallback ( void );
```

**Remark** \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**CSIF\textsubscript{n} ErrorCallback**

Performs processing in response to the CSIF\(n\) reception error interrupt INTCFnR (overrun error).

**Remark**
This API function is called as the callback routine of interrupt process MD_INTCFnR corresponding to the CSIF\(n\) reception error interrupt INTCFnR.

**[Classification]**

CG_serial_user.c

**[Syntax]**

```c
void CSIF\textsubscript{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 IIC0\(_n\)_Init ( void );
```

*Remark*  \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**IIC0\_UserInit**

Performs user-defined initialization relating to the IIC bus.

**Remark**  This API function is called as the \texttt{IIC0\_Init} callback routine.

**[Classification]**  
CG\_serial\_user.c

**[Syntax]**

```c
void IIC0\_UserInit ( void );
```

**Remark**  \( n \) is the channel number.

**[Argument(s)]**  
None.

**[Return value]**  
None.
IIC0n_Stop

Ends IIC0n communication.

[Classification]
CG_serial.c

[Syntax]

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

Remark  

\( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
Generates a stop condition.

[Classification]
CG_serial.c

[Syntax]

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

Remark  
$n$ is the channel number.

[Argument(s)]
None.

[Return value]
None.
**IIIC0n_MasterSendStart**

Starts IIIC0n master transmission.

**Remark**  This API function repeats the byte-level IIIC0n master transmission from the buffer specified in parameter *txbuf* the number of times specified in parameter *txnum*.

**[Classification]**

CG_serial.c

**[Syntax]**

```c
#include "CG_macrodriver.h"
MD_STATUS IIIC0n_MasterSendStart ( UCHAR adr, UCHAR *txbuf, USHORT txnum, UCHAR wait );
```

**Remark**  *n* is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>UCHAR *adr;</td>
<td>Device address</td>
</tr>
<tr>
<td>I</td>
<td>UCHAR *txbuf;</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>USHORT txnum;</td>
<td>Total amount of data to send</td>
</tr>
<tr>
<td>I</td>
<td>UCHAR wait;</td>
<td>Setup time of start conditions</td>
</tr>
</tbody>
</table>

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

![Device address diagram](image)

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ERROR</td>
<td>Exit with error (abend)</td>
</tr>
</tbody>
</table>
**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]**

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

**Remark**  `n` is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>UCHAR</td>
<td>adr;</td>
</tr>
<tr>
<td>O</td>
<td>UCHAR</td>
<td>rxbuf;</td>
</tr>
<tr>
<td>I</td>
<td>USHORT</td>
<td>rxnum;</td>
</tr>
<tr>
<td>I</td>
<td>UCHAR</td>
<td>wait;</td>
</tr>
</tbody>
</table>

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

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ERROR</td>
<td>Exit with error (abend)</td>
</tr>
</tbody>
</table>
**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]**

```c
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]**

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

**Remark**  \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**IIC0\_n\_MasterErrorCallback**

Performs processing in response to detection of error in IIC0\_n master communication.

**Remark**  This API function is called as the callback routine (process carried out when an IIC0\_n master communication error is detected) of interrupt process MD\_INTIIC\_n corresponding to the IIC0\_n master transfer complete interrupt INTIIC\_n.

**[Classification]**

CG\_serial\_user.c

**[Syntax]**

```c
#include "CG_macrodriver.h"
void IIC0\_n\_MasterErrorCallback ( MD\_STATUS flag );
```

**Remark**  \( n \) is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>MD_STATUS flag;</td>
<td>Cause of IIC0_n master communication error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_SPT: Stop condition detected</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_NACK: Acknowledge detected</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
**IIIC0\_SlaveSendStart**

Starts IIIC0 slave transmission.

**Remark**  This API function repeats the byte-level IIIC0 slave transmission from the buffer specified in parameter `txbuf` the number of times specified in parameter `txnum`.

**[Classification]**
CG\_serial.c

**[Syntax]**

```c
#include "CG\_macrodriver.h"
void IIIC0\_SlaveSendStart ( UCHAR *txbuf, USHORT txnum );
```

**Remark**  `n` is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>UCHAR *txbuf;</td>
<td>Pointer to a buffer storing the transmission data</td>
</tr>
<tr>
<td>I</td>
<td>USHORT txnum;</td>
<td>Total amount of data to send</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
**IIC0\_SlaveReceiveStart**

Starts IIC0\_n slave reception.

**Remark**  This API function performs byte-level IIC0\_n 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]**

```c
#include "CG_macrodriver.h"

void IIC0\_SlaveReceiveStart ( UCHAR *rxbuf, USHORT rxnum );
```

**Remark**  \( n \) is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>UCHAR *rxbuf;</td>
<td>Pointer to a buffer to store the received data</td>
</tr>
<tr>
<td>I</td>
<td>USHORT rxnum;</td>
<td>Total amount of data to receive</td>
</tr>
</tbody>
</table>

**[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]

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

**Remark**  n is the channel number.

[Argument(s)]

None.

[Return value]

None.
**IIC0\_n\_SlaveReceiveEndCallback**

Performs processing in response to the IIC0\_n slave transfer completion interrupt INTIIC\_n.

**Remark**  This API function is called as the callback routine of interrupt process MD\_INTIIC\_n corresponding to the simple IIC0\_n slave transfer completion interrupt INTIIC\_n (performed when total number of simple IIC0\_n slave transmissions specified by IIC0\_n\_SlaveReceiveStart parameter \textit{rxnum} has been completed).

**[Classification]**
CG\_serial\_user.c

**[Syntax]**

```c
void IIC0\_n\_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]**

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

**Remark**  n is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>MD_STATUS flag;</td>
<td>Cause of IIC0n master communication error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_ERROR: Address mismatch detected</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MD_NACK: Acknowledge detected</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
IIC0\_n\_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\_INTIIC\_n corresponding to the IIC0\_n transfer completion interrupt INTIIC\_n.

**[Classification]**
CG\_serial\_user.c

**[Syntax]**

```c
void IIC0\_n\_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>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AD_Init</strong></td>
<td>Performs initialization necessary to control A/D converter functions.</td>
</tr>
<tr>
<td><strong>AD_UserInit</strong></td>
<td>Performs user-defined initialization relating to the A/D converter.</td>
</tr>
<tr>
<td><strong>AD_Start</strong></td>
<td>Starts A/D conversion.</td>
</tr>
<tr>
<td><strong>AD_Stop</strong></td>
<td>Ends A/D conversion.</td>
</tr>
<tr>
<td><strong>AD_SelectADChannel</strong></td>
<td>Configures the analog voltage input pin for A/D conversion.</td>
</tr>
<tr>
<td><strong>AD_SetPFTCondition</strong></td>
<td>Sets the information for operation in power-fail compare mode (comparison value and A/D conversion end interrupt INTAD trigger).</td>
</tr>
<tr>
<td><strong>AD_Read</strong></td>
<td>Reads the results of A/D conversion (10 bits).</td>
</tr>
<tr>
<td><strong>AD_ReadByte</strong></td>
<td>Reads the results of A/D conversion (8 bits; most significant 8 bits of 10-bit resolution).</td>
</tr>
</tbody>
</table>
**AD_Init**

Performs initialization necessary to control A/D converter functions.

**[Classification]**

CG_ad.c

**[Syntax]**

```c
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]

```c
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]**

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

**[Argument(s)]**

None.

**[Return value]**

None.

**[Example]**

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

**[CG_main.c]**

```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]**

```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]**

```c
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]**

```c
#include "CG_macrodriver.h"
#include "CG_ad.h"

MD_STATUS AD_SelectADChannel ( enum ADChannel channel );
```

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>enum ADChannel channel;</td>
<td>Analog voltage input pin ADCHANNELn: Input pin</td>
</tr>
</tbody>
</table>

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

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
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)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>UCHAR pftvalue;</td>
<td>Comparison value</td>
</tr>
<tr>
<td>I</td>
<td>enum ADPFTMode mode;</td>
<td>Cause of A/D conversion end interrupt INTAD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EACHEND: Generate INTAD when A/D is complete</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PFTHIGHER: Generate INTAD if ADA0CRnH &gt; ADA0PFT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PFTLOWER: Generate INTAD if ADA0CRnH &lt; ADA0PFT</td>
</tr>
</tbody>
</table>

[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 registernH (ADA0CRnH).

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
AD_Read

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

[Classification]
CG_ad.c

[Syntax]

```c
#include "CG_macrodriver.h"

MD_STATUS AD_Read ( USHORT *buffer );
```

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>USHORT *buffer</td>
<td>Pointer to area in which to store read results of A/D conversion (10 bits)</td>
</tr>
</tbody>
</table>

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

![Results of A/D conversion to be stored in buffer]

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ERROR</td>
<td>Exit with error (abend)</td>
</tr>
</tbody>
</table>
**AD_ReadByte**

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

**[Classification]**

CG_ad.c

**[Syntax]**

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

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>UCHAR *buffer;</td>
<td></td>
</tr>
</tbody>
</table>

- 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`.

```
<table>
<thead>
<tr>
<th></th>
<th>B10</th>
<th>B9</th>
<th>B8</th>
<th>B7</th>
<th>B6</th>
<th>B5</th>
<th>B4</th>
<th>B3</th>
<th>B2</th>
<th>B1</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Results of A/D conversion to be stored in `buffer`

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ERROR</td>
<td>Exit with error (abend)</td>
</tr>
</tbody>
</table>
C.3.7 D/A Converter

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAn_Init</td>
<td>Performs initialization necessary to control D/A converter functions.</td>
</tr>
<tr>
<td>DAn_UserInit</td>
<td>Performs user-defined initialization relating to the D/A converter.</td>
</tr>
<tr>
<td>DAn_Start</td>
<td>Starts D/A conversion.</td>
</tr>
<tr>
<td>DAn_Stop</td>
<td>Ends D/A conversion.</td>
</tr>
<tr>
<td>DAn_SetValue</td>
<td>Sets the analog voltage output to the ANOn pin.</td>
</tr>
</tbody>
</table>
**DA\(_n\)_Init**

Performs initialization necessary to control D/A converter functions.

[Classification]
CG_da.c

[Syntax]

```c
void DA\(_n\)_Init ( void );
```

Remark  
\( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
**DA\textsubscript{n} UserInit**

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

**Remark**  This API function is called as the \texttt{DA\textsubscript{n} Init} callback routine.

**[Classification]**  
CG\_da\_user.c

**[Syntax]**

```c
void    DA\textsubscript{n} UserInit ( void );
```

**Remark**  \(n\) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
DA\(_n\)_Start

Starts D/A conversion.

[Classification]
CG_da.c

[Syntax]

\[
\text{void} \quad \text{DA}_n\_\text{Start} ( \text{void} );
\]

Remark \( n \) is the channel number.

[Argument(s)]
None.

[Return value]
None.
DA\_n\_Stop

Ends D/A conversion.

[Classification]
CG\_da\_c

[Syntax]

\[
\text{void DA}_n\_Stop ( \text{void} );
\]

Remark \( n \) is the channel number.

[Argument(s)]
None.

[Return value]
None.
**DA\_n\_SetValue**

Sets the analog voltage output to the ANOn pin.

**[Classification]**

CG\_da\_c

**[Syntax]**

```c
#include    "CG_macrodriver.h"
void    DA\_n\_SetValue ( UCHAR value );
```

**Remark**  
$n$ is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>UCHAR value;</td>
<td>Analog voltage (0x0 to 0xff)</td>
</tr>
</tbody>
</table>

**[Return value]**

None.

**[Example]**

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

**[CG\_main\_c]**

```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]**

```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>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMPn_Init</td>
<td>Performs initialization necessary to control 16-bit timer/event counter P (TMP) functions.</td>
</tr>
<tr>
<td>TMPn_UserInit</td>
<td>Performs user-defined initialization relating to the 16-bit timer/event counter P (TMP).</td>
</tr>
<tr>
<td>TMPn_Start</td>
<td>Starts the count for 16-bit timer/event counter P (TMP).</td>
</tr>
<tr>
<td>TMPn_Stop</td>
<td>Ends the count for 16-bit timer/event counter P (TMP).</td>
</tr>
<tr>
<td>TMPn_ChangeTimerCondition</td>
<td>Changes the counter value for 16-bit timer/event counter P (TMP).</td>
</tr>
<tr>
<td>TMPn_GetPulseWidth</td>
<td>Reads the pulse width of 16-bit timer/event counter P (TMP) (high/low level width).</td>
</tr>
<tr>
<td>TMPn_GetFreeRunningValue</td>
<td>Reads the value captured by 16-bit timer/event counter P (TMP).</td>
</tr>
<tr>
<td>TMPn_ChangeDuty</td>
<td>Changes the duty ratio of the PWM signal.</td>
</tr>
<tr>
<td>TMPn_SoftwareTriggerOn</td>
<td>Generates the trigger (software trigger) for timer output.</td>
</tr>
<tr>
<td>TMQ0_Init</td>
<td>Performs initialization necessary to control 16-bit timer/event counter Q (TMQ) functions.</td>
</tr>
<tr>
<td>TMQ0_UserInit</td>
<td>Performs user-defined initialization relating to the 16-bit timer/event counter Q (TMQ).</td>
</tr>
<tr>
<td>TMQ0_Start</td>
<td>Starts the count for 16-bit timer/event counter Q (TMQ).</td>
</tr>
<tr>
<td>TMQ0_Stop</td>
<td>Ends the count for 16-bit timer/event counter Q (TMQ).</td>
</tr>
<tr>
<td>TMQ0_ChangeTimerCondition</td>
<td>Changes the counter value for 16-bit timer/event counter Q (TMQ).</td>
</tr>
<tr>
<td>TMQ0_GetPulseWidth</td>
<td>Reads the pulse width of 16-bit timer/event counter Q (TMQ) (high/low level width).</td>
</tr>
<tr>
<td>TMQ0_GetFreeRunningValue</td>
<td>Reads the value captured by 16-bit timer/event counter Q (TMQ).</td>
</tr>
<tr>
<td>TMQ0_ChangeDuty</td>
<td>Changes the duty ratio of the PWM signal.</td>
</tr>
<tr>
<td>TMQ0_SoftwareTriggerOn</td>
<td>Generates the trigger (software trigger) for timer output.</td>
</tr>
<tr>
<td>TAAn_Init</td>
<td>Performs initialization necessary to control 16-bit timer/event counter AA (TAA) functions.</td>
</tr>
<tr>
<td>TAAn_UserInit</td>
<td>Performs user-defined initialization relating to the 16-bit timer/event counter AA (TAA).</td>
</tr>
<tr>
<td>TAAn_Start</td>
<td>Starts the count for 16-bit timer/event counter AA (TAA).</td>
</tr>
<tr>
<td>TAAn_Stop</td>
<td>Ends the count for 16-bit timer/event counter AA (TAA).</td>
</tr>
<tr>
<td>TAAn_ChangeTimerCondition</td>
<td>Changes the counter value for 16-bit timer/event counter AA (TAA).</td>
</tr>
<tr>
<td>TAAn_ControlOutputToggle</td>
<td>Changes the toggle control of 16-bit timer/event counter AA (TAA).</td>
</tr>
<tr>
<td>TAAn_GetPulseWidth</td>
<td>Reads the pulse width of 16-bit timer/event counter AA (TAA) (high/low level width).</td>
</tr>
<tr>
<td>TAAn_GetFreeRunningValue</td>
<td>Reads the value captured by 16-bit timer/event counter AA (TAA).</td>
</tr>
<tr>
<td>TAAn_ChangeDuty</td>
<td>Changes the duty ratio of the PWM signal.</td>
</tr>
<tr>
<td>TAAn_SoftwareTriggerOn</td>
<td>Generates the trigger (software trigger) for timer output.</td>
</tr>
<tr>
<td>TABn_Init</td>
<td>Performs initialization necessary to control 16-bit timer/event counter AB (TAB) functions.</td>
</tr>
<tr>
<td>TABn_UserInit</td>
<td>Performs user-defined initialization relating to the 16-bit timer/event counter AB (TAB).</td>
</tr>
<tr>
<td>TABn_Start</td>
<td>Starts the count for 16-bit timer/event counter AB (TAB).</td>
</tr>
<tr>
<td>API Function Name</td>
<td>Function</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TABn_Stop</td>
<td>Ends the count for 16-bit timer/event counter AB (TAB).</td>
</tr>
<tr>
<td>TABn_ChangeTimerCondition</td>
<td>Changes the counter value for 16-bit timer/event counter AB (TAB).</td>
</tr>
<tr>
<td>TABn_ControlOutputToggle</td>
<td>Changes the toggle control of 16-bit timer/event counter AB (TAB).</td>
</tr>
<tr>
<td>TABn_GetPulseWidth</td>
<td>Reads the pulse width of 16-bit timer/event counter AB (TAB) (high/low level width).</td>
</tr>
<tr>
<td>TABn_GetFreeRunningValue</td>
<td>Reads the value captured by 16-bit timer/event counter AB (TAB).</td>
</tr>
<tr>
<td>TABn_ChangeDuty</td>
<td>Changes the duty ratio of the PWM signal.</td>
</tr>
<tr>
<td>TABn_SoftwareTriggerOn</td>
<td>Generates the trigger (software trigger) for timer output.</td>
</tr>
<tr>
<td>TMT0_Init</td>
<td>Performs initialization necessary to control 16-bit timer/event counter T (TMT) functions.</td>
</tr>
<tr>
<td>TMT0_UserInit</td>
<td>Performs user-defined initialization relating to the 16-bit timer/event counter T (TMT).</td>
</tr>
<tr>
<td>TMT0_Start</td>
<td>Starts the count for 16-bit timer/event counter T (TMT).</td>
</tr>
<tr>
<td>TMT0_Stop</td>
<td>Ends the count for 16-bit timer/event counter T (TMT).</td>
</tr>
<tr>
<td>TMT0_ChangeTimerCondition</td>
<td>Changes the counter value for 16-bit timer/event counter T (TMT).</td>
</tr>
<tr>
<td>TMT0_GetPulseWidth</td>
<td>Reads the pulse width of 16-bit timer/event counter T (TMT) (high/low level width).</td>
</tr>
<tr>
<td>TMT0_GetFreeRunningValue</td>
<td>Reads the value captured by 16-bit timer/event counter T (TMT).</td>
</tr>
<tr>
<td>TMT0_ChangeDuty</td>
<td>Changes the duty ratio of the PWM signal.</td>
</tr>
<tr>
<td>TMT0_SoftwareTriggerOn</td>
<td>Generates the trigger (software trigger) for timer output.</td>
</tr>
<tr>
<td>TMT0_EnableHold</td>
<td>Changes the encoder counter control of 16-bit timer/event counter T (TMT) to holding count value.</td>
</tr>
<tr>
<td>TMT0_DisableHold</td>
<td>Changes the encoder counter control of 16-bit timer/event counter T (TMT) to normal operation.</td>
</tr>
<tr>
<td>TMT0_ChangeCountValue</td>
<td>Changes the initial counter value for 16-bit timer/event counter T (TMT).</td>
</tr>
<tr>
<td>TMMn_Init</td>
<td>Performs initialization necessary to control 16-bit interval timer M (TMM) functions.</td>
</tr>
<tr>
<td>TMMn_UserInit</td>
<td>Performs user-defined initialization relating to the 16-bit interval timer M (TMM).</td>
</tr>
<tr>
<td>TMMn_Start</td>
<td>Starts the count for 16-bit interval timer M (TMM).</td>
</tr>
<tr>
<td>TMMn_Stop</td>
<td>Ends the count for 16-bit interval timer M (TMM).</td>
</tr>
<tr>
<td>TMMn_ChangeTimerCondition</td>
<td>Changes the counter value for 16-bit interval timer M (TMM).</td>
</tr>
</tbody>
</table>
**TMP\textsubscript{n} Init**

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

**[Classification]**

CG\_timer.c

**[Syntax]**

```c
void TMP\textsubscript{n} Init ( void );
```

**Remark**  
\(n\) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**TMPn_UserInit**

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

**Remark**  
This API function is called as the **TMPn_Init** callback routine.

**[Classification]**  
CG_timer_user.c

**[Syntax]**

```c
void TMPn_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 TMP\(_n\)\_Start ( void );
```

**Remark** \(n\) is the channel number.

**[Argument(s)]**
- None.

**[Return value]**
- None.
**TMPn_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]**

```c
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 `arrar_reg` is set in TMPn capture/compare register m (TPnCCRm).

### [Classification]

CG_timer.c

### [Syntax]

```c
#include "CG_macrodriver.h"

MD_STATUS TMPn_ChangeTimerCondition ( USHORT *array_reg, UCHAR array_num );
```

**Remark**  `n` is the channel number.

### [Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>USHORT *array_reg;</td>
<td>Pointer to an area storing the count value (0x0 to 0xffff)</td>
</tr>
</tbody>
</table>
| I   | UCHAR array_num; | Register to change  
|     |                | 1: TPnCCR0  
|     |                | 2: TPnCCR0, TPnCCR1 |

### [Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>

### [Example]

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

```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 */
......
}


**TMP\textsubscript{n} 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 TMP\textsubscript{n} GetPulseWidth ( ULONG *activewith, ULONG *inactivewidth );
```

Remark \( n \) is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>ULONG *activewith;</td>
<td>Pointer to an area storing the high level width that was read (0x0 to 0x1ffff)</td>
</tr>
<tr>
<td>O</td>
<td>ULONG *inactivewidth;</td>
<td>Pointer to an area storing the low level width that was read (0x0 to 0x1ffff)</td>
</tr>
</tbody>
</table>

**[Return value]**
None.
**TMP\_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 (TPnCCRm) is selected as the capture register.

[**Classification**]
CG\_timer.c

[**Syntax**]

```c
#include "CG_macrodriver.h"
#include "CG_timer.h"
MD\_STATUS TMP\_n\_GetFreeRunningValue ( ULONG *count, enum TMChannel channel );
```

**Remark**  \(n\) is the channel number.

[**Argument(s)**]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>ULONG *count;</td>
<td>Pointer to area in which to store the width that was read</td>
</tr>
<tr>
<td>I</td>
<td>enum TMChannel channel;</td>
<td>Channel to read</td>
</tr>
</tbody>
</table>

**Macro Description**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>

---

**Remark**  \(n\) is the channel number.
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]**

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

**Remark**  n is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>UCHAR array_duty</td>
<td>Duty ratio (0 to 100, unit: %)</td>
</tr>
</tbody>
</table>

**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]**

```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 */
    ......
}
```
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]**

```c
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]

```c
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]**

```c
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]**

```c
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]**

```c
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"TQ0CCRm"`.

**[Classification]**

CG_timer.c

**[Syntax]**

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

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>USHORT *array_reg;</td>
<td>Pointer to an area storing the count value (0x0 to 0xffff)</td>
</tr>
<tr>
<td>I</td>
<td>UCHAR array_num;</td>
<td>Register to change</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: TQ0CCR0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2: TQ0CCR0, TQ0CCR1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3: TQ0CCR0, TQ0CCR1, TQ0CCR2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4: TQ0CCR0, TQ0CCR1, TQ0CCR2, TQ0CCR3</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>

**[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]**

```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_ChannelTimerCondition ( &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]**

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

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>ULONG *activewith;</td>
<td>Pointer to an area storing the high level width that was read (0x0 to 0x1fff)</td>
</tr>
<tr>
<td>O</td>
<td>ULONG *inactivewidth;</td>
<td>Pointer to an area storing the low level width that was read (0x0 to 0x1fff)</td>
</tr>
</tbody>
</table>

**[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 (TQ0CCRm) is selected as the capture register.

**[Classification]**

CG_timer.c

**[Syntax]**

```c
#include "CG_macrodriver.h"
#include "CG_timer.h"

MD_STATUS TMQ0_GetFreeRunningValue ( ULONG *count, enum TMChannel channel );
```

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>ULONG *count;</td>
<td>Pointer to area in which to store the width that was read</td>
</tr>
<tr>
<td>I</td>
<td>enum TMChannel channel;</td>
<td>Channel to read</td>
</tr>
</tbody>
</table>

- TMCHANNEL0: Channel 0 (TQ0CCR0)
- TMCHANNEL1: Channel 1 (TQ0CCR1)
- TMCHANNEL2: Channel 2 (TQ0CCR2)
- TMCHANNEL3: Channel 3 (TQ0CCR3)

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
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]

```c
#include "CG_macrodriver.h"

MD_STATUS TMQ0_ChangeDuty ( UCHAR *array_duty, UCHAR array_num );
```

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>UCHAR *array_duty;</td>
<td>Pointer to an area storing the duty ratio (0 to 100; in percent)</td>
</tr>
<tr>
<td>I</td>
<td>UCHAR array_num;</td>
<td>Register to change</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: TQ0CCR1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2: TQ0CCR1, TQ0CCR2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3: TQ0CCR1, TQ0CCR2, TQ0CCR3</td>
</tr>
</tbody>
</table>

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

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>

[Example]

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

[CG_main.c]

```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]**

```c
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]

```c
void TAA\_n\_Init ( void );
```

Remark  \( n \) is the channel number.

[Argument(s)]
None.

[Return value]
None.
TAA\textsubscript{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\textsubscript{n} Init callback routine.

[Classification]

CG\_timer\_user.c

[Syntax]

\begin{verbatim}
void TAA\textsubscript{n} UserInit ( void );
\end{verbatim}

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]**

```c
void TAA\_n\_Start ( void );
```

**Remark**  \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
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]**

```c
void TAA_n_Stop ( void );
```

**Remark**  $n$ is the channel number.

**[Argument(s)]**
- None.

**[Return value]**
- None.
**TAA\textsubscript{n} ChangeTimerCondition**

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

**Remark** The value specified in parameter *array\_reg* is set in TAA\textsubscript{n} capture/compare register m (TAA\textsubscript{n}CCR\textsubscript{m}).

**[Classification]**

CG\_timer.c

**[Syntax]**

```c
#include    "CG_macrodriver.h"
MD\_STATUS TAA\textsubscript{n} ChangeTimerCondition ( USHORT *array\_reg, UCHAR array\_num );
```

**Remark** \( n \) is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>USHORT *array_reg;</td>
<td>Pointer to an area storing the count value (0x0 to 0xffff)</td>
</tr>
<tr>
<td>I</td>
<td>UCHAR array_num;</td>
<td>Register to change 1: TAA\textsubscript{n}CCR\textsubscript{0} 2: TAA\textsubscript{n}CCR\textsubscript{0}, TAA\textsubscript{n}CCR\textsubscript{1}</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
**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]**

```c
#include "CG\_macrodriver\_h"
#include "CG\_timer\_h"

MD\_STATUS TAA\_n\_ControlOutputToggle ( enum TMOut toggle, enum TMChannel channel );
```

**Remark** \( n \) is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>enum TMOut toggle</td>
<td>Toggle control</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STANDARD: Normal toggle operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INACTIVE: Reset request</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACTIVE: Set request</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FREEZE: Keep request</td>
</tr>
<tr>
<td>I</td>
<td>enum TMChannel channel</td>
<td>pin to change</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMCHANNEL0: TOAA_n_0 pin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMCHANNEL1: TOAA_n_1 pin</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
**TAA\textsubscript{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]**

```c
#include "CG_macrodriver.h"

void TAA\textsubscript{n} GetPulseWidth ( ULONG *activewith, ULONG *inactivewidth );
```

**Remark**  
\(n\) is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>ULONG *activewith;</td>
<td>Pointer to an area storing the high level width that was read (0x0 to 0x1fff)</td>
</tr>
<tr>
<td>O</td>
<td>ULONG *inactivewidth;</td>
<td>Pointer to an area storing the low level width that was read (0x0 to 0x1fff)</td>
</tr>
</tbody>
</table>

**[Return value]**

None.
**TAA\textsubscript{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\textsubscript{n} capture/compare register m (TAA\textsubscript{n}CCR\textsubscript{m}) is selected as the capture register.

**[Classification]**  
CG_timer.c

**[Syntax]**

```c
#include    "CG_macrodriver.h"
#include    "CG_timer.h"
MD_STATUS   TAA\textsubscript{n} GetFreeRunningValue ( ULONG *count, enum TMChannel channel );
```

**Remark**  \( n \) is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>ULONG *count;</td>
<td>Pointer to area in which to store the width that was read</td>
</tr>
<tr>
<td>I</td>
<td>enum TMChannel channel;</td>
<td>Channel to read</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMCHANNEL0:  Channel 0 (TAA\textsubscript{n}CCR\textsubscript{0})</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMCHANNEL1:  Channel 1 (TAA\textsubscript{n}CCR\textsubscript{1})</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
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]**

```c
#include    "CG_macrodriver.h"

void    TAA\textsubscript{n} \_ChangeDuty ( UCHAR array\_duty );
```

**Remark**  \( n \) is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>UCHAR array_duty;</td>
<td>Duty ratio (0 to 100, unit: %)</td>
</tr>
</tbody>
</table>

**Remark**  The value set to duty ratio \( array\_duty \) must be in base 10 notation.

**[Return value]**

None.
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]**

```c
void TAA\_n\_SoftwareTriggerOn ( void );
```

**Remark** \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**TABn_Init**

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

**[Classification]**

CG_timer.c

**[Syntax]**

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

**Remark**  

$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**TABn_UserInit**

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

**Remark**  This API function is called as the TABn_Init callback routine.

**[Classification]**

CG_timer_user.c

**[Syntax]**

```c
void TABn_UserInit ( void );
```

**Remark**  \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**TABn_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]**

```c
void TABn_Start ( void );
```

**Remark** \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**TABn_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]**

```c
void    TABn_Stop ( void );
```

**Remark** $n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**TABn_ChangeTimerCondition**

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

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

**[Classification]**

CG_timer.c

**[Syntax]**

```c
#include "CG_macrodriver.h"

MD_STATUS TABn_ChangeTimerCondition ( USHORT *array_reg, UCHAR array_num );
```

**Remark**  *n* is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>USHORT *array_reg;</td>
<td>Pointer to an area storing the count value (0x0 to 0xffff)</td>
</tr>
<tr>
<td>I</td>
<td>UCHAR array_num;</td>
<td>Register to change</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: TABnCCR0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2: TABnCCR0, TABnCCR1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3: TABnCCR0, TABnCCR1, TABnCCR2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4: TABnCCR0, TABnCCR1, TABnCCR2, TABnCCR3</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
**TABn_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]**

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

**Remark** $n$ is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>enum TMOutput toggle;</td>
<td>Toggle control</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STANDARD: Normal toggle operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INACTIVE: Reset request</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACTIVE: Set request</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FREEZE: Keep request</td>
</tr>
<tr>
<td></td>
<td>enum TMChannel channel;</td>
<td>pin to change</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMCHANNEL0: TOABn0 pin</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMCHANNEL1: TOABn1 pin</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
**TABn_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]**

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

**Remark**

- `n` is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>ULONG *activewith;</td>
<td>Pointer to an area storing the high level width that was read (0x0 to 0x1ffff)</td>
</tr>
<tr>
<td>O</td>
<td>ULONG *inactivewidth;</td>
<td>Pointer to an area storing the low level width that was read (0x0 to 0x1ffff)</td>
</tr>
</tbody>
</table>

**[Return value]**

- None.
**TABn_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 TABn capture/compare register m (TABnCCRm) is selected as the capture register.

**[Classification]**

CG_timer.c

**[Syntax]**

```c
#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)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>ULONG *count;</td>
<td>Pointer to area in which to store the width that was read</td>
</tr>
<tr>
<td>I</td>
<td>enum TMChannel channel;</td>
<td>Channel to read</td>
</tr>
</tbody>
</table>

- TMCHANNEL0: Channel 0 (TABnCCR0)
- TMCHANNEL1: Channel 1 (TABnCCR1)
- TMCHANNEL2: Channel 2 (TABnCCR2)
- TMCHANNEL3: Channel 3 (TABnCCR3)

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
**TABn_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]**

```c
#include "CG_macrodriver.h"

MD_ATATUS TABn_ChangeDuty ( UCHAR array_duty, UCHAR array_num );
```

**Remark**  *n* is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>UCHAR array_duty;</td>
<td>Duty ratio (0 to 100, unit: %)</td>
</tr>
<tr>
<td>I</td>
<td>UCHAR array_num;</td>
<td>Register to change</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: TABnCCR1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2: TABnCCR1, TPnCCR2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3: TABnCCR1, TPnCCR2, TPnCCR3</td>
</tr>
</tbody>
</table>

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

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
**TABn_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]**

```c
void TABn_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]**

```c
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]**

```c
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]

```c
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]

```c
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` (TT0CCRm).

### [Classification]

CG_timer.c

### [Syntax]

```c
#include "CG_macrodriver.h"

MD_STATUS TMT0_ChangeTimerCondition ( USHORT *array_reg, UCHAR array_num );
```

### [Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>USHORT *array_reg;</td>
<td>Pointer to an area storing the count value (0x0 to 0xffff)</td>
</tr>
<tr>
<td>I</td>
<td>UCHAR array_num;</td>
<td>Register to change</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: TT0CCR0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2: TT0CCR0, TT0CCR1</td>
</tr>
</tbody>
</table>

### [Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
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]**

```c
#include "CG_macrodriver.h"

void TMT0_GetPulseWidth ( ULONG *activewith, ULONG *inactivewidth );
```

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>ULONG *activewith;</td>
<td>Pointer to an area storing the high level width that was read (0x0 to 0x1fff)</td>
</tr>
<tr>
<td>O</td>
<td>ULONG *inactivewidth;</td>
<td>Pointer to an area storing the low level width that was read (0x0 to 0x1fff)</td>
</tr>
</tbody>
</table>

**[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 (TT0CCRm) is selected as the capture register.

**[Classification]**

CG_timer.c

**[Syntax]**

```c
#include "CG_macrodriver.h"
#include "CG_timer.h"

MD_STATUS TMT0_GetFreeRunningValue ( ULONG *count, enum TMChannel channel);
```

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>ULONG *count;</td>
<td>Pointer to area in which to store the width that was read</td>
</tr>
<tr>
<td>I</td>
<td>enum TMChannel channel;</td>
<td>Channel to read</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMCHANNEL0: Channel 0 (TPnCCR0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TMCHANNEL1: Channel 1 (TPnCCR1)</td>
</tr>
</tbody>
</table>

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
**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]**

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

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>UCHAR array_duty;</td>
<td>Duty ratio (0 to 100, unit: %)</td>
</tr>
</tbody>
</table>

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

**[[Return value]]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
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]**

```c
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]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
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]**
```c
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]
```c
#include "CG_macrodriver.h"

void TMT0_ChangeCountValue ( USHORT regvalue );
```

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>USHORT <code>regvalue</code>;</td>
<td>Count value (0x0 to 0xffff)</td>
</tr>
</tbody>
</table>

[Return value]
None.
**TMM\textsubscript{n} Init**

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

**[Classification]**

CG_timer.c

**[Syntax]**

```c
void TMM\textsubscript{n} Init ( void );
```

**Remark** \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**TMMn_UserInit**

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

**Remark**  This API function is called as the **TMMn_Init** callback routine.

**[Classification]**

CG_timer_user.c

**[Syntax]**

```c
void TMMn_UserInit ( void );
```

**Remark**  \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**TMMn_Start**

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

**[Classification]**

CG_timer.c

**[Syntax]**

```c
void TMMn_Start ( void );
```

**Remark**  

$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**TMMn_Stop**

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

**[Classification]**

CG_timer.c

**[Syntax]**

```c
void TMMn_Stop ( void );
```

**Remark**  
$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
CubeSuite+ V1.03.00

APPENDIX C  API FUNCTIONS

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]

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

Remark   n is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>USHORT regvalue;</td>
<td>Counter value (0x0 to 0xffff)</td>
</tr>
</tbody>
</table>

[Return value]

None.
C.3.9 Watch Timer

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

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>WT_Init</td>
<td>Performs initialization necessary to control watch timer functions.</td>
</tr>
<tr>
<td>WT_UserInit</td>
<td>Performs user-defined initialization relating to the watch timer.</td>
</tr>
<tr>
<td>WT_Start</td>
<td>Clears the watch timer counter and resumes counting.</td>
</tr>
<tr>
<td>WT_Stop</td>
<td>Ends the count for watch timer.</td>
</tr>
</tbody>
</table>
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]

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

[Argument(s)]
None.

[Return value]
None.
**WT_Stop**

Ends the count for watch timer.

**[Classification]**

CG_wt.c

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.

**[Example]**

The example below illustrates use of the watch timer function.

**[CG_main.c]**

```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]**

```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>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTC_Init</td>
<td>Performs initialization necessary to control real-time counter functions.</td>
</tr>
<tr>
<td>RTC_UserInit</td>
<td>Performs user-defined initialization relating to the real-time counter.</td>
</tr>
<tr>
<td>RTC_CounterEnable</td>
<td>Starts the count of the real-time counter (year, month, weekday, day, hour, minute, second).</td>
</tr>
<tr>
<td>RTC_CounterDisable</td>
<td>Ends the count of the real-time counter (year, month, weekday, day, hour, minute, second).</td>
</tr>
<tr>
<td>RTC_SetHourSystem</td>
<td>Sets the clock type (12-hour or 24-hour clock) of the real-time counter.</td>
</tr>
<tr>
<td>RTC_CounterSet</td>
<td>Sets the counter value (year, month, weekday, day, hour, minute, second) of the real-time counter.</td>
</tr>
<tr>
<td>RTC_CounterGet</td>
<td>Reads the counter value (year, month, weekday, day, hour, minute, second) of the real-time counter.</td>
</tr>
<tr>
<td>RTC_ConstPeriodInterruptEnable</td>
<td>Sets the cycle of the interrupts INTRTC0, then starts the cyclic interrupt function.</td>
</tr>
<tr>
<td>RTC_ConstPeriodInterruptDisable</td>
<td>Ends the cyclic interrupt function.</td>
</tr>
<tr>
<td>RTC_AlarmEnable</td>
<td>Starts the alarm interrupt function.</td>
</tr>
<tr>
<td>RTC_AlarmDisable</td>
<td>Ends the alarm interrupt function.</td>
</tr>
<tr>
<td>RTC_AlarmSet</td>
<td>Sets the alarm conditions (weekday, hour, minute).</td>
</tr>
<tr>
<td>RTC_AlarmGet</td>
<td>Reads the alarm conditions (weekday, hour, minute).</td>
</tr>
<tr>
<td>RTC_IntervalStart</td>
<td>Starts the interval interrupt function.</td>
</tr>
<tr>
<td>RTC_IntervalStop</td>
<td>Ends the interval interrupt function.</td>
</tr>
<tr>
<td>RTC_IntervalInterruptEnable</td>
<td>Sets the cycle of the interrupts INTRTC2, then starts the interval interrupt function.</td>
</tr>
<tr>
<td>RTC_IntervalInterruptDisable</td>
<td>Ends the interval interrupt function.</td>
</tr>
<tr>
<td>RTC_RC1CK1HZ_OutputEnable</td>
<td>Enables output of the real-time counter correction clock (1 Hz) to the RC1CK1HZ pin.</td>
</tr>
<tr>
<td>RTC_RC1CK1HZ_OutputDisable</td>
<td>Disables output of the real-time counter correction clock (1 Hz) to the RC1CK1HZ pin.</td>
</tr>
<tr>
<td>RTC_RC1CKO_OutputEnable</td>
<td>Enables output of the real-time counter clock (32 kHz source) to the RC1CKO pin.</td>
</tr>
<tr>
<td>RTC_RC1CKO_OutputDisable</td>
<td>Disables output of the real-time counter clock (32 kHz source) to the RC1CKO pin.</td>
</tr>
<tr>
<td>RTC_RC1CKDIV_OutputEnable</td>
<td>Enables output of the real-time counter clock (32 kHz cycle) to the RC1CKDIV pin.</td>
</tr>
<tr>
<td>RTC_RC1CKDIV_OutputDisable</td>
<td>Disables output of the real-time counter clock (32 kHz cycle) to the RC1CKDIV pin.</td>
</tr>
<tr>
<td>RTC_RTC1HZ_OutputEnable</td>
<td>Enables output of the real-time counter correction clock (1 Hz) to the RTC1HZ pin.</td>
</tr>
<tr>
<td>RTC_RTC1HZ_OutputDisable</td>
<td>Disables output of the real-time counter correction clock (1 Hz) to the RTC1HZ pin.</td>
</tr>
<tr>
<td>RTC_RTCCCL_OutputEnable</td>
<td>Enables output of the real-time counter clock (32 kHz source) to the RTCCCL pin.</td>
</tr>
<tr>
<td>RTC_RTCCCL_OutputDisable</td>
<td>Disables output of the real-time counter clock (32 kHz source) to the RTCCCL pin.</td>
</tr>
<tr>
<td>RTC_RTCDIV_OutputEnable</td>
<td>Enables output of the real-time counter clock (32 kHz cycle) to the RTCDIV pin.</td>
</tr>
<tr>
<td>RTC_RTCDIV_OutputDisable</td>
<td>Disables output of the real-time counter clock (32 kHz cycle) to the RTCDIV pin.</td>
</tr>
<tr>
<td>RTC_changeCorrectionValue</td>
<td>Changes the timing and correction value for correcting clock errors.</td>
</tr>
</tbody>
</table>
RTC_Init

Performs initialization necessary to control real-time counter functions.

[Classification]
CG rtc.c

[Syntax]

```c
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]

```c
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]

```c
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]

```c
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]

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

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>enum RTCHourSystem hoursystem;</td>
</tr>
</tbody>
</table>

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_BUSY1</td>
<td>Executing count process (before change to setting)</td>
</tr>
<tr>
<td>MD_BUSY2</td>
<td>Stopping count process (after change to setting)</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>

[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.

```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]**

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

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>struct RTCCounterValue counterwriteval;</td>
<td>Counter value</td>
</tr>
</tbody>
</table>

**Remark**  Below is an example of the structure RTCCounterValue (counter value) for the real-time counter.

```c
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]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_BUSY1</td>
<td>Executing count process (before change to setting)</td>
</tr>
<tr>
<td>MD_BUSY2</td>
<td>Stopping count process (after change to setting)</td>
</tr>
</tbody>
</table>

**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".
```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]**

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

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>struct RTCounterValue *counterreadval;</td>
<td>Pointer to structure in which to store the counter value being read</td>
</tr>
</tbody>
</table>

**Remark**  See **RTC_CounterSet** for details about the RTCounterValue counter value.

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_BUSY1</td>
<td>Executing count process (before reading)</td>
</tr>
<tr>
<td>MD_BUSY2</td>
<td>Stopping count process (after reading)</td>
</tr>
</tbody>
</table>

**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]**

```c
#include "CG_macrodriver.h"
#include "CG_rtc.h"

void main ( void ) {
    struct RTCounterValue 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

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

#### Argument(s)

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>enum RTCINTPeriod period;</td>
<td>Interrupt INTRTC0 cycle</td>
</tr>
</tbody>
</table>

  - HALFSEC: 0.5 seconds
  - ONESEC: 1 second
  - ONEMIN: 1 minute
  - ONEHOUR: 1 hour
  - ONEDAY: 1 day
  - ONEMONTH: 1 month

#### Return value

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specfication</td>
</tr>
</tbody>
</table>

#### Example

Below is an example of setting the cycle of the interrupts INTRTC0, then starting the cyclic interrupt function.

**[CG_main.c]**

```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]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
RTC_AlarmEnable

Starts the alarm interrupt function.

[Classification]
CG_rtc.c

[Syntax]

```c
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]

```c
#include    "CG_macrodriver.h"
#include    "CG_rtc.h"

void    RTC_AlarmSet ( struct RTCAlarmValue alarmval );
```

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>struct</td>
<td>Alarm conditions (weekday, hour, minute)</td>
</tr>
<tr>
<td></td>
<td>RTCAlarmValue</td>
<td>alarmval;</td>
</tr>
</tbody>
</table>

[Remark]
Below is shown the structure RTCAlarmValue (alarm conditions).

```c
struct  RTCAlarmValue {
    UCHAR   Alarmwm;    /* Minute */
    UCHAR   Alarmwh;    /* Hour */
    UCHAR   Alarmww;    /* Weekday */
};
```

- **Alarmwm (Minute)**

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

  ![Diagram of Alarmwm (Minute)]

  - BCD code (minute: digit 10)
  - BCD code (minute: digit 1)
  - 0: Fixed

- **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]
```c
#include "CG_macrodriver.h"
#include "CG_rtc.h"

void main ( void ) {
    struct RTCAlarmValue alarmval;
    ......
    RTC_AlarmEnable ();       /* Start alarm interrupt function */
    RTC.CounterEnable ();     /* Start count */
    ......
    RTC_SetHourSystem ( HOUR24 );       /* Set clock type */
    alarmval.Alarmww = 0xe;
    alarmval.Alarmwh = 0x17;
    alarmval.Alarmwm = 0x30;
    RTC.AlarmSet ( alarmval );         /* Set conditions */
    ......
}
```
[Example 2]

The example below shows the alarm conditions being set to "Saturday/Sunday (time left unchanged)".

[CG_main.c]

```c
#include "CG_macrodriver.h"
#include "CG_rtc.h"

void main ( void ) {
    struct RTCAlarmValue 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]

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

Remark  See RTC_AlarmSet for details about RTCAlarmValue (alarm conditions).

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>struct RTCAlarmValue *alarmval;</td>
<td>Pointer to structure in which to store the conditions being read</td>
</tr>
</tbody>
</table>

[Return value]

None.

[Example]

The example below shows the alarm conditions being read.

[CG_main.c]

```c
#include    "CG_macrodriver.h"
#include    "CG_rtc.h"
void main ( void ) {
    struct RTCAlarmValue 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]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
**RTC_IntervalStop**

Ends the interval interrupt function.

[Classification]

CG_rtc.c

[Syntax]

```c
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]

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

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>enum RTCINTInterval interval;</td>
<td>Interrupt INTRTC2 cycle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INTERVAL0: 2^6/fRTC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INTERVAL1: 2^7/fRTC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INTERVAL2: 2^8/fRTC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INTERVAL3: 2^9/fRTC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INTERVAL4: 2^10/fRTC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INTERVAL5: 2^11/fRTC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INTERVAL6: 2^12/fRTC</td>
</tr>
</tbody>
</table>

Remark  fRTC is the frequency of the subsystem clock.

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>

[Example]

Below is an example of changing the interval, the restarting the interval interrupt function.

[CG_main.c]

```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]

```c
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]

```c
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]

```c
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]

```c
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]

```c
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]

```c
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]

```c
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]

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

[Argument(s)]
None.

[Return value]
None.
RTC_RTCCL_OutputEnable

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

[Classification]
CG_rtc.c

[Syntax]

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

[Argument(s)]
None.

[Return value]
None.
RTC_RTCCL_OutputDisable

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

[Classification]
CG_rtc.c

[Syntax]
void RTC_RTCCL_OutputDisable ( void );

[Argument(s)]
None.

[Return value]
None.
RTC_RTCDIV_OutputEnable

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

[Classification]
CG_rtc.c

[Syntax]

```c
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]**

```c
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]**

```c
#include "CG_macrodriver.h"
#include "CG_rtc.h"

MD_STATUS RTC_ChangeCorrectionValue ( enum RTCCorectionTiming timing, UCHAR correctVal );
```

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>enum RTCCorectionTiming</td>
<td>When clock errors are corrected</td>
</tr>
<tr>
<td>I</td>
<td>timing;</td>
<td>EVERY20S: When the seconds digits are 00, 20 or 40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EVERY60S: When the seconds digits are 00</td>
</tr>
<tr>
<td>I</td>
<td>UCHAR correctVal;</td>
<td>Clock error correction value</td>
</tr>
</tbody>
</table>

**Remark**  
This API function does not correct clock errors if correction value `correctVal` is set to 0x0, 0x1, 0x40 or 0x41.

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
### C.3.11 Real-Time Output

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

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

Performs initialization necessary to control real-time output functions.

**[Classification]**

CG_rto.c

**[Syntax]**

```c
void RToN_Init ( void );
```

**Remark**  
$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**RTO\_UserInit**

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

**Remark**  This API function is called as the **RTO\_Init** callback routine.

**[Classification]**

CG\_rto\_user\_c

**[Syntax]**

```c
void RTO\_UserInit ( void );
```

**Remark**  \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
RTO_Enable

Enables (validates) real-time output.

[Classification]
CG_rto.c

[Syntax]

```c
void RTO_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]**

```c
void RTO\_n\_CounterDisable ( void );
```

**Remark**

$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
Sets 2-bit data for real-time output.

[Classification]
CG_rto.c

[Syntax]

```c
#include "CG_macrodriver.h"

void RTO_n_Set2BitData ( UCHAR data );
```

Remark  

\( n \) is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>UCHAR data;</td>
<td>2-bit data</td>
</tr>
</tbody>
</table>

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

[Return value]

None.
Sets 4-bit data for real-time output.

[Classification]

CG_rto.c

[Syntax]

```c
#include "CG_macrodriver.h"

void RTO_n_Set4BitsData ( UCHAR data );
```

Remark   \( n \) is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>UCHAR data;</td>
<td>4-bit data</td>
</tr>
</tbody>
</table>

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

[Return value]

None.
Sets 6-bit data for real-time output.

[Classification]
CG_rto.c

[Syntax]

```c
#include "CG_macrodriver.h"
void RTO[n]_Set6BitsData ( UCHAR data );
```

Remark  
\( n \) is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>UCHAR data;</td>
<td>6-bit data</td>
</tr>
</tbody>
</table>

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

[Return value]

None.
Sets 8-bit data for real-time output.

[Classification]
CG_rto.c

[Syntax]

```c
#include "CG_macrodriver.h"
void RTO\_n\_Set8BitsData ( UCHAR data );
```

Remark  \( n \) is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>UCHAR data;</td>
<td>8-bit data</td>
</tr>
</tbody>
</table>

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

[Return value]
None.
Sets higher 2-bit data for real-time output.

[Classification]
CG_rto.c

[Syntax]

```c
#include "CG_macrodriver.h"
void RTO_n_SetHigh2BitData ( UCHAR data );
```

Remark   \( n \) is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>UCHAR data;</td>
<td>Higher 2-bit data</td>
</tr>
</tbody>
</table>

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

```
\[
\begin{array}{cccccccc}
7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
\hline \\
B7 & B6 & B5 & B4 & B3 & B2 & B1 & B0 \\
\end{array}
\]

Higher 2-bit data

[Return value]
None.
**RTO\textsubscript{n}_SetLow2BitData**

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

**[Classification]**

CG\_rto\_c

**[Syntax]**

```c
#include "CG_macrodriver.h"

void RTO\textsubscript{n}_SetLow2BitsData ( UCHAR data );
```

*Remark*  \( n \) is the channel number.

**[Argument(s)]**

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>UCHAR data;</td>
<td>Lower 2-bit data</td>
</tr>
</tbody>
</table>

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

![Diagram showing 8-bit data with lower 2-bit data highlighted](image)

**[Return value]**

None.
Sets higher 4-bit data for real-time output.

[Classification]
CG_rto.c

[Syntax]

```c
#include "CG_macrodriver.h"
void RTO\_SetHigh4BitData ( UCHAR data );
```

Remark  
$n$ is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>UCHAR</td>
<td>data</td>
</tr>
</tbody>
</table>

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

[Return value]
None.
Sets lower 4-bit data for real-time output.

[Classification]
CG_rto.c

[Syntax]

```c
#include    "CG_macrodriver.h"
void    RTO_n_SetLow4BitData ( UCHAR data );
```

Remark  \( n \) is the channel number.

[Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>UCHAR data;</td>
<td>Lower 4-bit data</td>
</tr>
</tbody>
</table>

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

[Return value]

None.
**RTO\textsubscript{n} GetValue**

Reads data from real-time output.

**[Classification]**

CG\_rto.c

**[Syntax]**

```c
#include "CG_macrodriver.h"
void RTO\textsubscript{n} GetValue ( UCHAR *value );
```

**Remark**  \( n \) is the channel number.

**[Argument(s)]**


<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>UCHAR *value;</td>
<td>Pointer to area in which to store the value that was read</td>
</tr>
</tbody>
</table>

**[Return value]**

None.

**[Example]**

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

**[CG\_main.c]**

```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]**

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

Table C-13. API Functions: [DMA]

<table>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMAn_Init</td>
<td>Performs initialization necessary to control DMA controller functions.</td>
</tr>
<tr>
<td>DMAn_UserInit</td>
<td>Performs user-defined initialization relating to the DMA controller.</td>
</tr>
<tr>
<td>DMAn_Enable</td>
<td>Enables operation of channel ( n ).</td>
</tr>
<tr>
<td>DMAn_Disable</td>
<td>Disables operation of channel ( n ).</td>
</tr>
<tr>
<td>DMAn_CheckStatus</td>
<td>Reads the transfer status (transfer complete/transfer ongoing).</td>
</tr>
<tr>
<td>DMAn_SetData</td>
<td>Sets the RAM address of the transfer source/destination, and the number of times the data has been transferred.</td>
</tr>
<tr>
<td>DMAn_SoftwareTriggerOn</td>
<td>Uses a software trigger as a DMA transfer start trigger.</td>
</tr>
</tbody>
</table>
**DMAn_Init**

Performs initialization necessary to control DMA controller functions.

**[Classification]**

CG_dma.c

**[Syntax]**

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

**Remark**  
$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**DMAn_UserInit**

Performs user-defined initialization relating to the DMA controller.

**Remark**  This API function is called as the **DMAn_Init** callback routine.

**[Classification]**

CG_dma_user.c

**[Syntax]**

```c
void DMAn_UserInit ( void );
```

**Remark**  \( n \) is the channel number.

**[Argument(s)]**

None.

**[Return value]**

None.
**DMAn_Enable**

Enables operation of channel \( n \).

[Classification]

CG_dma.c

[Syntax]

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

**Remark**  
\( n \) is the channel number.

[Argument(s)]

None.

[Return value]

None.
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 \texttt{DMA\_CheckStatus}.

[Classification]
CG\_dma\_c

[Syntax]

\begin{verbatim}
void DMA\_Disable ( void );
\end{verbatim}

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]

\begin{verbatim}
#include "CG\_macrodriver\_h"

void main ( void ) {
    ......
    while ( MD\_COMPLETED == DMA0\_CheckStatus () ); /* Check transfer status */
    DMA0\_Disable (); /* Change to operation disabled status */
    ......
}
\end{verbatim}
**DMAn_CheckStatus**

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

**[Classification]**

CG_dma.c

**[Syntax]**

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

**Remark**  
$n$ is the channel number.

**[Argument(s)]**

None.

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_UNDEREXEC</td>
<td>Transfer ongoing</td>
</tr>
<tr>
<td>MD_COMPLETED</td>
<td>Transfer complete</td>
</tr>
</tbody>
</table>

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]  

```c
#include "CG_macrodriver.h"
MD_STATUS DMA\_n\_SetData ( UINT srcaddr, UINT dstaddr, UINT count );
```

**Remark** $n$ is the channel number.

---

### [Argument(s)]

<table>
<thead>
<tr>
<th>I/O</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>UINT srcaddr;</td>
<td>RAM address of source</td>
</tr>
<tr>
<td>I</td>
<td>UINT dstaddr;</td>
<td>RAM address of destination</td>
</tr>
<tr>
<td>I</td>
<td>UINT count;</td>
<td>Number of data transmissions (1 to 1024)</td>
</tr>
</tbody>
</table>

---

### [Return value]  

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Invalid argument specification</td>
</tr>
</tbody>
</table>
**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]**

```c
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]**

```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>
<thead>
<tr>
<th>API Function Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>LVI_Init</td>
<td>Performs initialization necessary to control low-voltage detector functions.</td>
</tr>
<tr>
<td>LVI_UserInit</td>
<td>Performs user-defined initialization relating to the low-voltage detector.</td>
</tr>
<tr>
<td>LVI_InterruptModeStart</td>
<td>Starts low-voltage detection (when in interrupt generation mode).</td>
</tr>
<tr>
<td>LVI_ResetModeStart</td>
<td>Starts low-voltage detection (when in internal reset mode).</td>
</tr>
<tr>
<td>LVI_Start</td>
<td>Starts low-voltage detection.</td>
</tr>
<tr>
<td>LVI_Stop</td>
<td>Stops low-voltage detection.</td>
</tr>
</tbody>
</table>
LVI_Init

Performs initialization necessary to control low-voltage detector functions.

[Classification]
CG_lvi.c

[Syntax]

```c
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]**

```c
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]

```c
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]

```c
void main ( void ) {
    ......
    LVI_InterruptModeStart ( ); /* Start low-voltage detection */
    ......
}
```

[CG_lvi_user.c]

```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]**

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

**[Argument(s)]**

None.

**[Return value]**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ERROR</td>
<td>Exit with error (abend)</td>
</tr>
<tr>
<td></td>
<td>- The program is configured to not use the low-voltage detector function.</td>
</tr>
<tr>
<td></td>
<td>- The object of low voltage detection is external voltage (VDD), and power voltage (VDD) &lt;= detected voltage (VLVI).</td>
</tr>
<tr>
<td></td>
<td>- The object of low voltage detection is external input voltage (EXLVI), and external input voltage (EXLVI) &lt;= detected voltage (VEXLVI).</td>
</tr>
</tbody>
</table>
LVI_Start

Starts low-voltage detection.

[Classification]
CG_lvi.c

[Syntax]

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

[Argument(s)]
None.

[Return value]

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD_OK</td>
<td>Normal completion</td>
</tr>
<tr>
<td>MD_ARGERROR</td>
<td>Exit with error (abend)</td>
</tr>
<tr>
<td></td>
<td>- The program is configured to not use the low-voltage detector function.</td>
</tr>
<tr>
<td></td>
<td>- The object of low voltage detection is external voltage (VDD), and power voltage (VDD) &lt;= detected voltage (VLVI).</td>
</tr>
<tr>
<td></td>
<td>- The object of low voltage detection is external input voltage (EXLVI), and external input voltage (EXLVI) &lt;= detected voltage (VEXLVI).</td>
</tr>
</tbody>
</table>
**LVI_Stop**

Stops low-voltage detection.

**[Classification]**

CG_lvi.c

**[Syntax]**

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

**[Argument(s)]**

None.

**[Return value]**

None.
## APPENDIX D  INDEX

### A

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

### C

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CG_ChangeClockMode</td>
<td>106</td>
</tr>
<tr>
<td>CG_ChangeFrequency</td>
<td>107</td>
</tr>
<tr>
<td>CG_ReadResetSource</td>
<td>105</td>
</tr>
<tr>
<td>CG_SelectPllMode</td>
<td>110</td>
</tr>
<tr>
<td>CG_SelectPowerSaveMode</td>
<td>108</td>
</tr>
<tr>
<td>CG_SelectSSCGMode</td>
<td>111</td>
</tr>
<tr>
<td>CG_SelectStabTime</td>
<td>109</td>
</tr>
<tr>
<td>CLOCK_Init</td>
<td>103</td>
</tr>
<tr>
<td>CLOCK_UserInit</td>
<td>104</td>
</tr>
<tr>
<td>Code Generator panel</td>
<td>69</td>
</tr>
<tr>
<td>Code Generator Preview panel</td>
<td>72</td>
</tr>
<tr>
<td>Column Chooser dialog box</td>
<td>77</td>
</tr>
<tr>
<td>CRC_GetResult</td>
<td>115</td>
</tr>
<tr>
<td>CRC_SetData</td>
<td>114</td>
</tr>
<tr>
<td>CRC_Start</td>
<td>113</td>
</tr>
<tr>
<td>CSIb_ErrorCallback</td>
<td>180</td>
</tr>
<tr>
<td>CSIb_Init</td>
<td>171</td>
</tr>
<tr>
<td>CSIb_ReceiveData</td>
<td>176</td>
</tr>
<tr>
<td>CSIb_ReceiveEndCallback</td>
<td>179</td>
</tr>
<tr>
<td>CSIb_SendData</td>
<td>175</td>
</tr>
<tr>
<td>CSIb_SendEndCallback</td>
<td>178</td>
</tr>
<tr>
<td>CSIb_SendReceiveData</td>
<td>177</td>
</tr>
<tr>
<td>CSIEn_ErrorCallback</td>
<td>190</td>
</tr>
<tr>
<td>CSIEn_Init</td>
<td>181</td>
</tr>
<tr>
<td>CSIEn_ReceiveData</td>
<td>186</td>
</tr>
<tr>
<td>CSIEn_ReceiveEndCallback</td>
<td>189</td>
</tr>
<tr>
<td>CSIEn_SendData</td>
<td>185</td>
</tr>
<tr>
<td>CSIEn_SendEndCallback</td>
<td>188</td>
</tr>
<tr>
<td>CSIEn_SendReceiveData</td>
<td>187</td>
</tr>
<tr>
<td>CSIEn_Start</td>
<td>183</td>
</tr>
<tr>
<td>CSIEn_Stop</td>
<td>184</td>
</tr>
<tr>
<td>CSIEn_UserInit</td>
<td>182</td>
</tr>
<tr>
<td>CSIFn_ErrorCallback</td>
<td>200</td>
</tr>
<tr>
<td>CSIFn_Init</td>
<td>191</td>
</tr>
</tbody>
</table>
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
  DMAn_CheckStatus ... 352
  DMAn_Disable ... 351
  DMAn_Enable ... 350
  DMAn_Init ... 348
  DMAn_SetData ... 353
  DMAn_SoftwareTriggerOn ... 354
  DMAn_UserInit ... 349

DMAn_SoftwareTriggerOn ... 354
DMAn_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
  IICOn_GetStopConditionCallback ... 215
  IICOn_Init ... 201
  IICOn_MasterErrorCallback ... 209
  IICOn_MasterReceiveEndCallback ... 208
  IICOn_MasterReceiveStart ... 206
  IICOn_MasterSendEndCallback ... 207
  IICOn_MasterSendStart ... 205
  IICOn_SlaveErrorCallback ... 214
  IICOn_SlaveReceiveEndCallback ... 213
  IICOn_SlaveReceiveStart ... 211
  IICOn_SlaveSendEndCallback ... 212
  IICOn_SlaveSendStart ... 210
  IICOn_Stop ... 203
  IICOn_StopCondition ... 204
  IICOn_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
RTOn_Disable ... 337
RTOn_Enable ... 336
RTOn_GetValue ... 346
RTOn_Init ... 334
RTOn_Set2BitData ... 338
RTOn_Set4BitData ... 339
RTOn_Set6BitData ... 340
RTOn_Set8BitData ... 341
RTOn_SetHigh2BitData ... 342
RTOn_SetHigh4BitData ... 344
RTOn_SetLow2BitData ... 343
RTOn_SetLow4BitData ... 345
RTOn_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.IntervalInterruptDisable ... 319
RTC.IntervalInterruptEnable ... 317
RTC.IntervalStart ... 315
RTC.IntervalStop ... 316
RTC_RC1CK1HZ_OutputDisable ... 321
RTC_RC1CK1HZ_OutputEnable ... 320
RTC_RC1CKDIV_OutputDisable ... 325
RTC_RC1CKDIV_OutputEnable ... 324
RTC_RTC1HZ_OutputDisable ... 327
RTC_RTC1HZ_OutputEnable ... 326
RTC_RTCCL_OutputDisable ... 329
RTC_RTCCL_OutputEnable ... 328
RTC_RTCDIV_OutputDisable ... 330
RTC_RTCDIV_OutputEnable ... 331
RTC_RTCDIV_OutputDisable ... 330
RTC_RTCDIV_OutputEnable ... 331
RTC_SetHourSystem ... 301
RTC_UserInit ... 298
RTOn_Disable ... 337
RTOn_Enable ... 336
RTOn_GetValue ... 346
RTOn_Init ... 334
RTOn_Set2BitData ... 338
RTOn_Set4BitData ... 339
RTOn_Set6BitData ... 340
RTOn_Set8BitData ... 341
RTOn_SetHigh2BitData ... 342
RTOn_SetHigh4BitData ... 344
RTOn_SetLow2BitData ... 343
RTOn_SetLow4BitData ... 345
RTOn_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.IntervalInterruptDisable ... 319
RTC.IntervalInterruptEnable ... 317
RTC.IntervalStart ... 315
RTC.IntervalStop ... 316
RTC_RC1CK1HZ_OutputDisable ... 321
RTC_RC1CK1HZ_OutputEnable ... 320
RTC_RC1CKDIV_OutputDisable ... 325
RTC_RC1CKDIV_OutputEnable ... 324
RTC_RTC1HZ_OutputDisable ... 327
RTC_RTC1HZ_OutputEnable ... 326
RTC_RTCCL_OutputDisable ... 329
RTC_RTCCL_OutputEnable ... 328
RTC_RTCDIV_OutputDisable ... 330
RTC_RTCDIV_OutputEnable ... 331
RTC_RTCDIV_OutputDisable ... 330
RTC_RTCDIV_OutputEnable ... 331
RTC_SetHourSystem ... 301
RTC_UserInit ... 298
RTOn_Disable ... 337
RTOn_Enable ... 336
RTOn_GetValue ... 346
RTOn_Init ... 334
RTOn_Set2BitData ... 338
RTOn_Set4BitData ... 339
RTOn_Set6BitData ... 340
RTOn_Set8BitData ... 341
RTOn_SetHigh2BitData ... 342
RTOn_SetHigh4BitData ... 344
RTOn_SetLow2BitData ... 343
RTOn_SetLow4BitData ... 345
RTOn_UserInit ... 335
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
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
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_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>SelectPllMode ... 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
## APPENDIX D  INDEX

**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

**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

**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

**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
<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>Sep 01, 2012</td>
<td>- First Edition issued</td>
</tr>
</tbody>
</table>