User's Manual

RX62G Group

Peripheral Driver Generator
Reference Manual

Renesas Electronics
www.renesas.com

Rev.1.02 May 2014

Notice

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

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

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

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

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

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

Introduction

This manual was written to explain how to make the peripheral I/O drivers on the Peripheral Driver Generator for RX62G. For the basic information about the Peripheral Driver Generator, refer to the Peripheral Driver Generator user's manual.

Table of Contents

1. Overview

1.1 Supported peripheral modules

The Peripheral Driver Generator supports the following products of RX62G group, peripheral modules and endian.

(1) Products

Part No.	Package	Part No.	Package
R5F562GAADFH	PLQP0112JA-A	R5F562GADDFH	PLQP0112JA-A
R5F562GAADFP	PLQP0100KB-A	R5F562GADDFP	PLQP0100KB-A
R5F562G7ADFH	PLQP0112JA-A	R5F562G7DDFH	PLQP0112JA-A
R5F562G7ADFP	PLQP0100KB-A	R5F562G7DDFP	PLQP0100KB-A

(2) Peripheral Modules

Voltage Detection Circuit (LVD)	Compare Match Timer (CMT)
Clock Generation Circuit	Watchdog Timer (WDT)
Low Power Consumption	Independent Watchdog Timer (IWDT)
Interrupt Control Unit (ICU), Exceptions	Serial Communications Interface (SCIb)
Buses *Illegal address access detection	CRC Calculator (CRC)
Data Transfer Controller (DTC)	I2C Bus Interface (RIIC)
I/O Ports	Serial Peripheral Interface (RSPI)
Multi-Function Timer Pulse Unit 3 (MTU3)	LIN Module (LIN)
Port Output Enable 3 (POE3)	12-Bit A/D Converter (S12ADA)
General PWM Timer (GPTa)	10-Bit A/D Converter (ADA)

The CAN module is not supported.

The memory protection unit (MPU) is not supported.

The comparators in 12-Bit A/D converter (S12ADA) are not supported.

(3) Endian

Big endian, Little endian

1.2 Tool requirements

The following tools are required for this version of RX262G group Peripheral Driver Generator.

- RX Family C/C++ Compiler Package V.1.02 Release 01
- RX62G/RX62T Group Renesas Peripheral Driver Library V.1.10 (Bundled in Peripheral Driver Generator)

2. Creating a new project

To create the new project file, select the menu [File] -> [New Project]. New project dialog box will open.

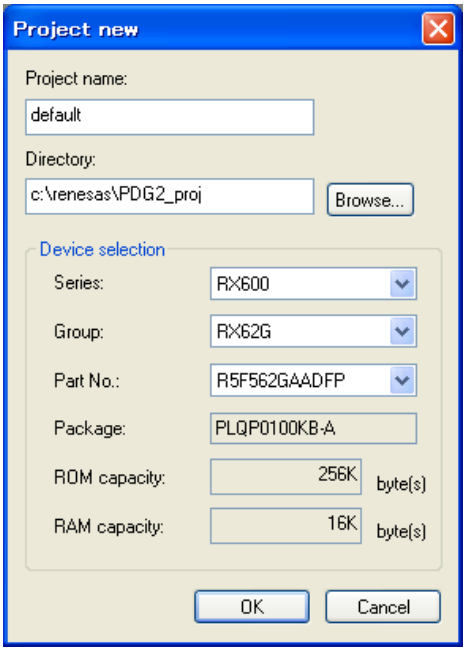


Fig 2.1 New project dialog box

For RX62G group, select [RX600] as a series and select [RX62G] as a group. The package type, ROM capacity and RAM capacity of selected product are displayed.

By clicking [OK], new project is created and opened.

The EXTAL input clock frequency is not set after opening a new project. Therefore an error icon is displayed. For error display, refer to the user's manual.

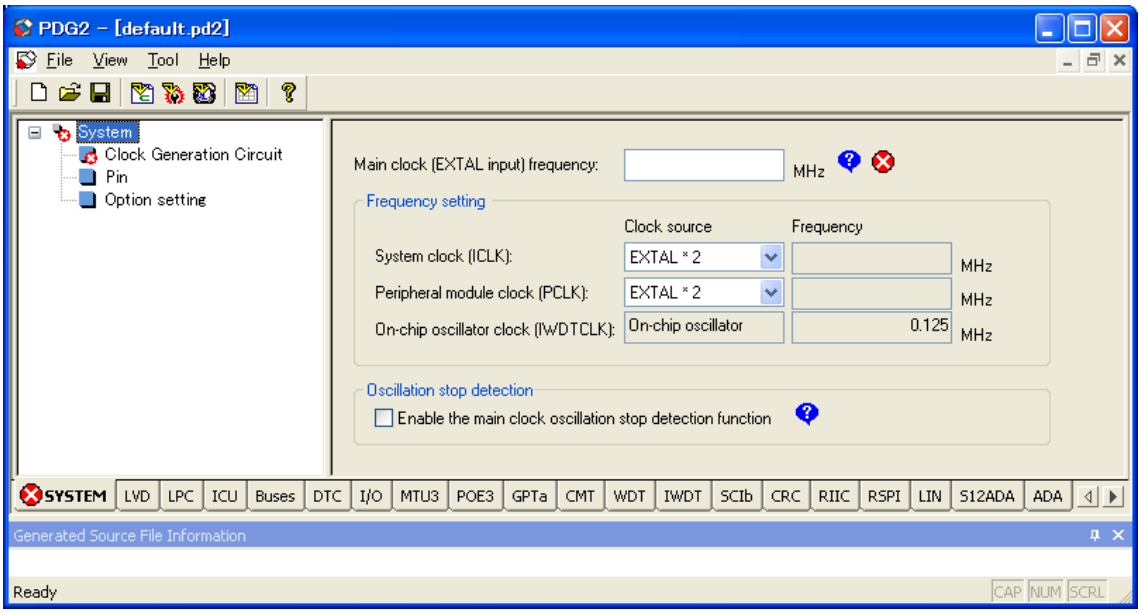


Fig 2.2 Error display of new project

Set the frequency of the lock to be used here.

3. Setting Up the Peripheral Modules

3.1 Main Window

Figure 3.1 shows the main window for setting up peripheral modules.

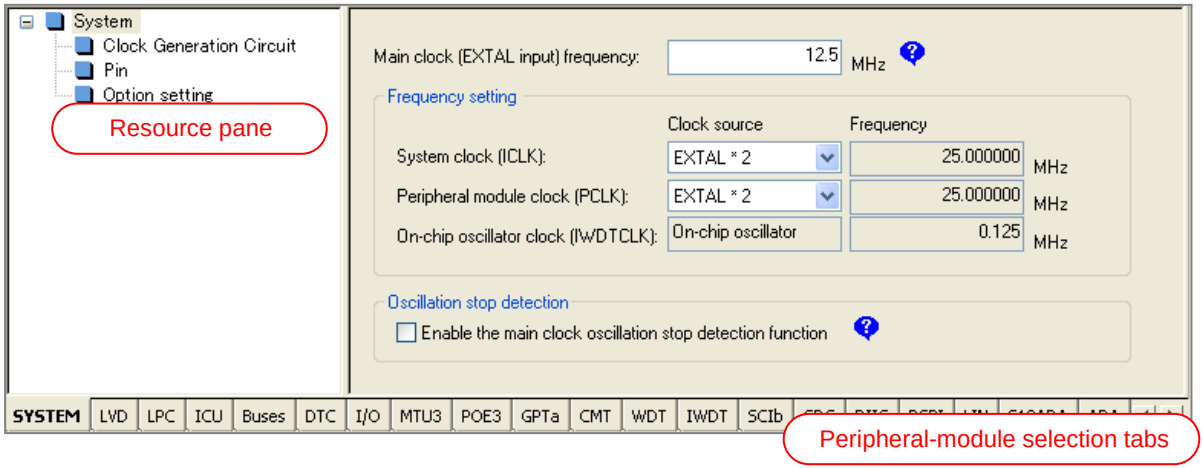


Figure 3.1 Display in the Main Window (Example)

Table 3.1 shows the correspondence between the peripheral-module selection tabs, items in the resource pane, and peripheral modules to be set up.

Table 3.1 Peripheral-Module Selection Tabs, Items in the Resource Pane, and Peripheral Modules

Tab	Resource pane	Corresponding Peripheral Module or Function
SYSTEM	Clock Generation Circuit	Clock Generation Circuit
	Pin	Pinfunctions
LVD	LVD	LVD1 and 2
LPC	Low Power Consumption	Low Power Consumption Functions
ICU	Interrupts	Interrupt Control Unit (ICU) (Fastinterrupt, Software Interrupt, External Interrupt (NMI, IRQ0 to IRQ7))
	Exceptions	Exceptions
Buses	Bus Error Monitoring	Bus Error Monitoring (Illegal address access detection)
DTC	DTC	Data Transfer Controller (DTC)
I/O	Port 0 to Port G	I/O Port 0 to G
MTU3	MTU3_0 to MTU3_7	Multi-Function Timer Pulse Unit 3 (MTU3) Channel 0 to 7
POE3	POE3	Port Output Enable 3 (POE3)
GPTa	GPT0 to GPT3	General PWM Timer (GPT) Channel 0 to 3
CMT	Unit0 (CMT0 and CMT1)	Compare Match Timer (CMT) Unit 0 (Channlel 0 and 1)
	Unit1 (CMT2 and CMT3)	Compare Match Timer (CMT) Unit 1 (Channlel 2 and 3)
WDT	WDT	Watchdog Timer
IWDT	IWDT	Independent Watchdog Timer
SCIb	SCI0 to SCI2	Serial Communications Interface (SCIb) Channel 0 to 2
CRC	CRC	CRC Calculator (CRC)
RIIC	RIIC0	I2C Bus Interface (RIIC) Channel 0
RSPI	RSPI0	Serial Peripheral Interface (RSPI) Channel 0
LIN	LIN0	LIN Module (LIN) Channel 0
S12ADA	S12ADA0 and S12ADA1	12-Bit A/D Converter (S12ADA) Unit 0 and 1
ADA	ADA0	10-Bit A/D Converter (ADA) Unit 0

For how to set up the peripheral modules, refer to the user's manual. For details on the setting of pin functions, refer to section 3.2, Pin Functions.

3.2 Pin Functions

Select the [SYSTEM] tab from the peripheral-module selection tabs and click on [Pin] in the resource pane to open the pin-function pane.

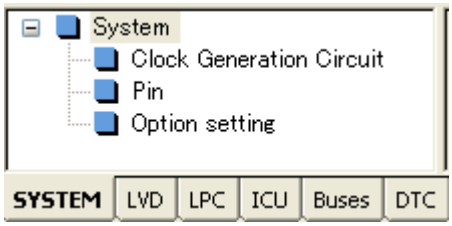


Figure 3.2 Opening the Pin-Function Pane

The pin-function pane has [Pin function] and [Peripheral pin usage] sheets.

3.2.1 [Pin function] Sheet

The [Pin function] sheet shows all of the MCU pins in order.

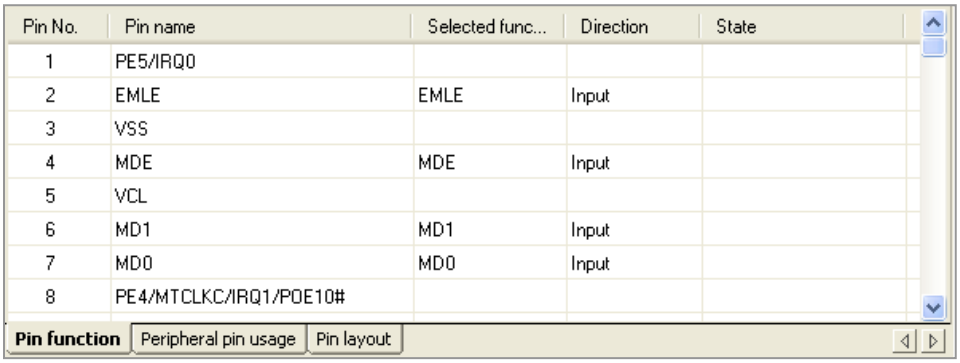


Figure 3.3 Pin-Function Pane ([Pin function] Sheet)

The contents of each column are shown in table 3.2.

Table 3.2 Columns on the [Pin function] Sheet

Column	Description
Pin No.	Pin number
Pin name	Name of the pin (which shows all of the functions assigned to that pin)
Selected function	Pin function for the selected peripheral module
Direction	Input or output
State	Current state

When a peripheral module associated with input to or output from pins has been set up, the current setting is shown on the [Pin function] sheet. In 112-pin LQFP package, if you have set A/D converter AD0 in the detailed settings pane up so that the input on analog input pin AN0 will be converted, for example, the [Pin function] sheet shows the setting of pin 86 (P60/AN0) as follows.

Pin No.	Pin name	Selected function	Direction	State
86	P60/AN0	AN0	Input	

Figre 3.4 Display of selected pin function

Setting up I/O port P60 in this state will cause a conflict between P60 and AN0 and a warning message will be output as shown in figure 3.5.

Pin No.	Pin name	Selected function	Direction	State
⚠ 86	P60/AN0	AN0/P60		Conflicting between different functions.

Figre 3.5 Confliction between Pin Functions

Notes

- Pin-by-pin designation of pin functions for RX62G-group MCUs is not possible because the settings of peripheral modules automatically determine the pin functions. The assigned pin functions also cannot be changed in this pane.
- The allocations of some pin functions, however, can be changed on the [Peripheral pin usage] sheet.
- If two or more output functions are enabled on a single pin, the pin only outputs the signal of the function with the highest priority. For details, refer to the hardware manual.

3.2.2 [Peripheral pin usage] Sheet

The [Peripheral pin usage] sheet shows which pins are used by the corresponding peripheral module. The pin functions specific to the peripheral module selected in the left section are listed in the right section.

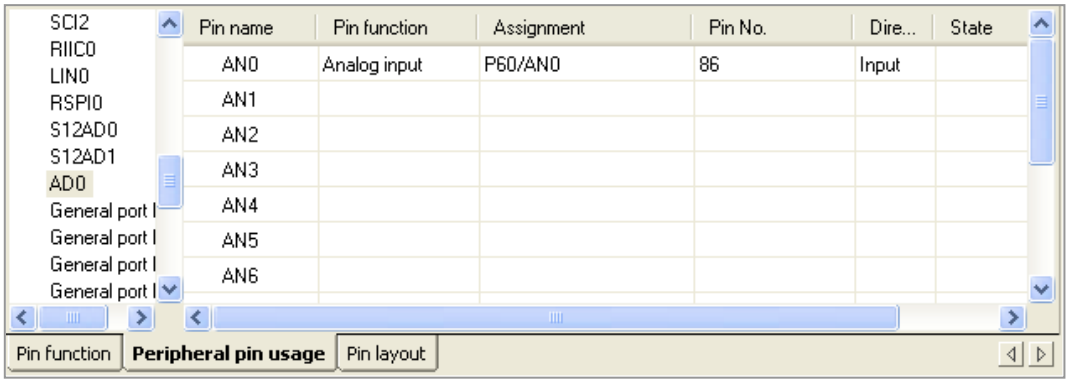


Figure 3.6 Pin-Function Pane ([Peripheral pin usage] Sheet)

Table 3.3 lists the columns on the [Peripheral pin usage] sheet.

Table 3.3 Columns on the [Peripheral pin usage] Sheet

Column	Contents
Pin Name	Names of pins used by the peripheral module selected in the left section
Pin Function	Pin function
Assignment	Full name of the MCU pin, showing all of the functions assigned to that pin
Pin No.	Pin number
Direction	Input or output
State	Current state

When a peripheral module associated with input to or output from pins has been set up, the current setting is shown on the [Peripheral pin usage] sheet. In 112-pin LQFP package, if you have set external interrupt IRQ0 in the detailed settings pane up, for example, the [Peripheral pin usage] sheet shows the setting of pin IRQ0 as follows.

Pin name	Pin function	Assignment	Pin No.	Direction	State
IRQ0	External interrupt	P10/MTCLKD/IRQ0	110	Input	

Figre 3.7 Display of a Pin Function (Example)

Then setting up I/O port P10, which uses the same pin as IRQ0, will cause a conflict between P10 and IRQ0 and a warning message will be output as shown in figure 3.8.

Pin name	Pin function	Assignment	Pin No.	Direction	State
⚠ IRQ0	External interrupt	P10/MTCLKD/IRQ0	110	Input	Conflicting with another pin function.

Figure 3.8 Confliction between pin functions

Other pins to which IRQ0 can be assigned are selectable from a drop-down list box. Placing the mouse pointer on the [Assignment] column brings up a drop-down button.

Pin name	Pin function	Assignment	Pin No.	Direction	State
⚠ IRQ0	External interrupt	P10/MTCLKD/IRQ	110	Input	Conflicting with another pin function.

Figure 3.9 Drop-Down Button

Click on the drop-down button and select one of the options displayed in the list box.

Pin name	Pin function	Assignment	Pin No.	Direction	State
⚠ IRQ0	External interrupt	P10/MTCLKD/IRQ	110	Input	Conflicting with another pin function.

P10/MTCLKD/IRQ0
PE5/IRQ0
PG0/IRQ0/TRSYNC

Figure 3.10 Changing the Allocation of a Pin Function

If IRQ0 is assigned to PE5/IRQ0 and that pin is not being used by any other peripheral module, the conflict between P10 and IRQ0 can be resolved.

Pin name	Pin function	Assignment	Pin No.	Direction	State
IRQ0	External interrupt	PE5/IRQ0	1	Input	

Figure 3.11 Display after Changing the Allocation

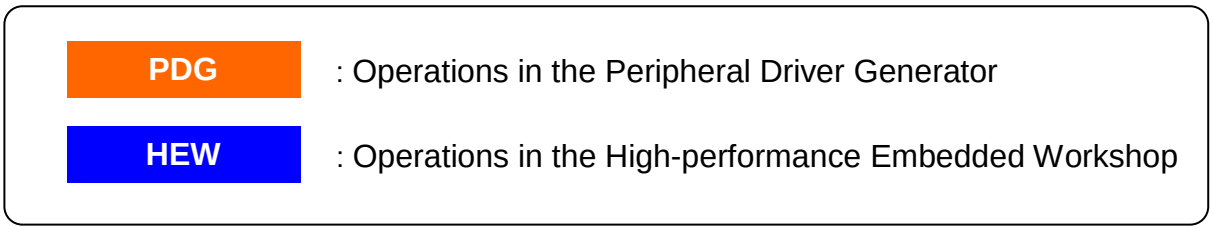
The pin functions for which you can select the assignment are listed in appendix 1, Pin Functions for which the Allocation Can be Changed.

4. Tutorial

This section introduces the usage of the Peripheral Driver Generator by giving instructions on how to use the Peripheral Driver Generator and High-performance Embedded Workshop to create a tutorial program that implements the following operations on the Renesas Starter Kit board for the RX62G.

- An LED blinking on a Compare Match Timer (CMT) interrupt
- An LED blinking on the PWM output of the multi-function timer pulse unit 3 (MTU3)
- Continuously scanning on 10-Bit A/D converter (ADA)
- Triggering DTC by IRQ
- Data transfer between SCIb channels 0 and 2

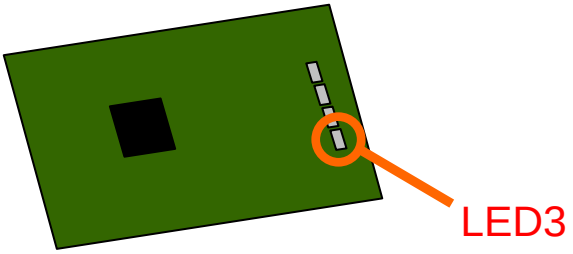
The labels given below respectively indicate operations to take place in the Peripheral Driver Generator and in the High-performance Embedded Workshop.



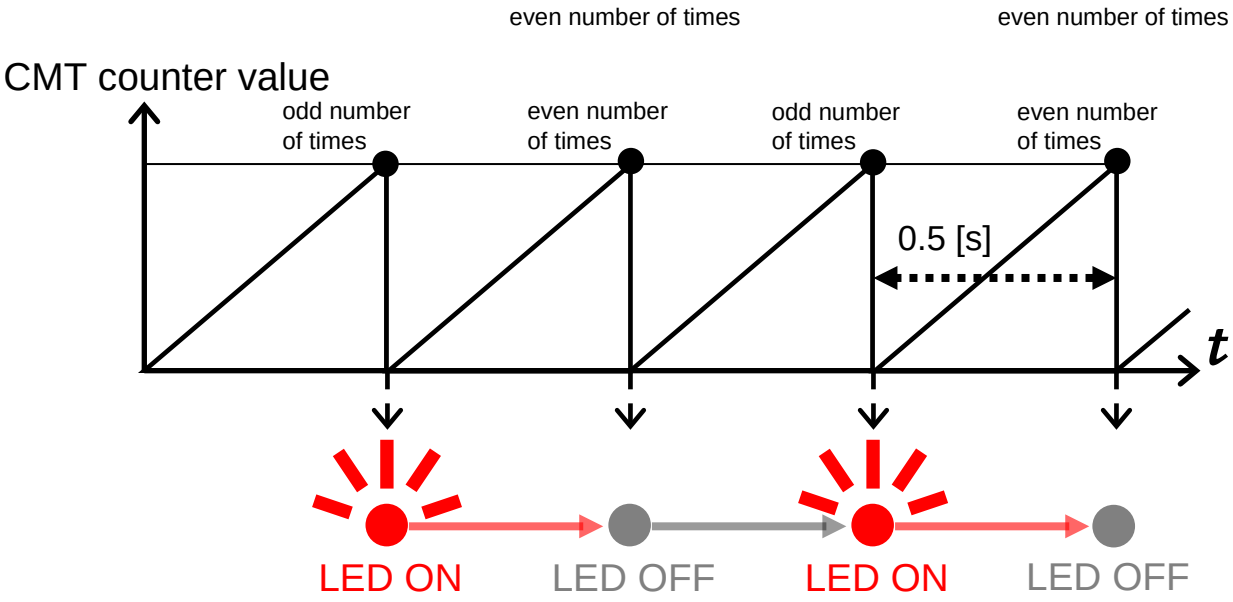
4.1 An LED blinking on a Compare Match Timer (CMT) interrupt

The LED3 on RSK board is connected to P33. In this tutorial, Compare Match Timer and I/O port will be set up to blink this LED as follows.

The LED3 turns on when the output from P33 is 0, and turns off when the output is 1.

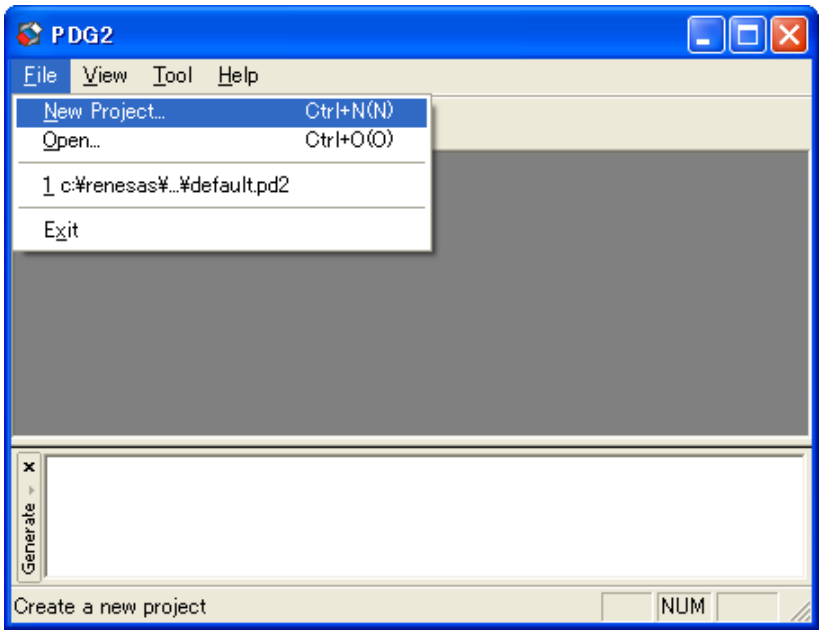


- Turn on the LED
  at compare match
  (odd number of times)
- Turn off the LED
  at compare match
  (even number of times)



(1) Making the Peripheral Driver Generator project **PDG**

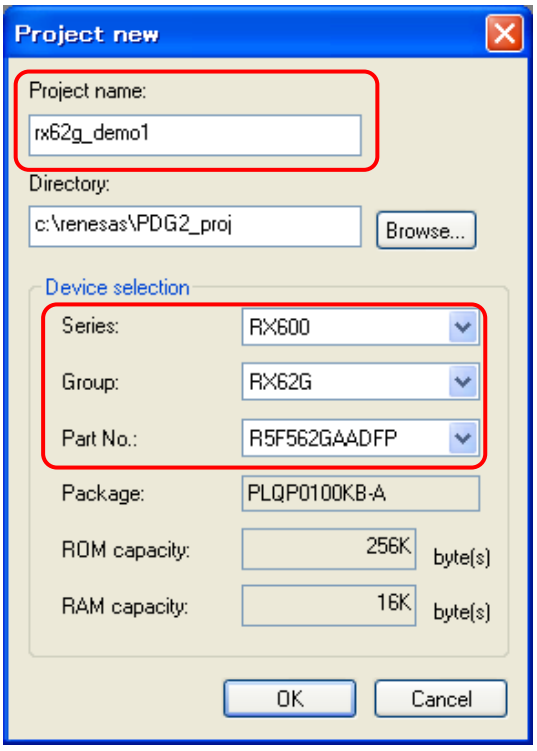
1. Start the Peripheral Driver Generator.
2. Select [File]->[New Project] menu.



3. Specify "rx62g_demo1" as the project name.
   Set the CPU type as follows.

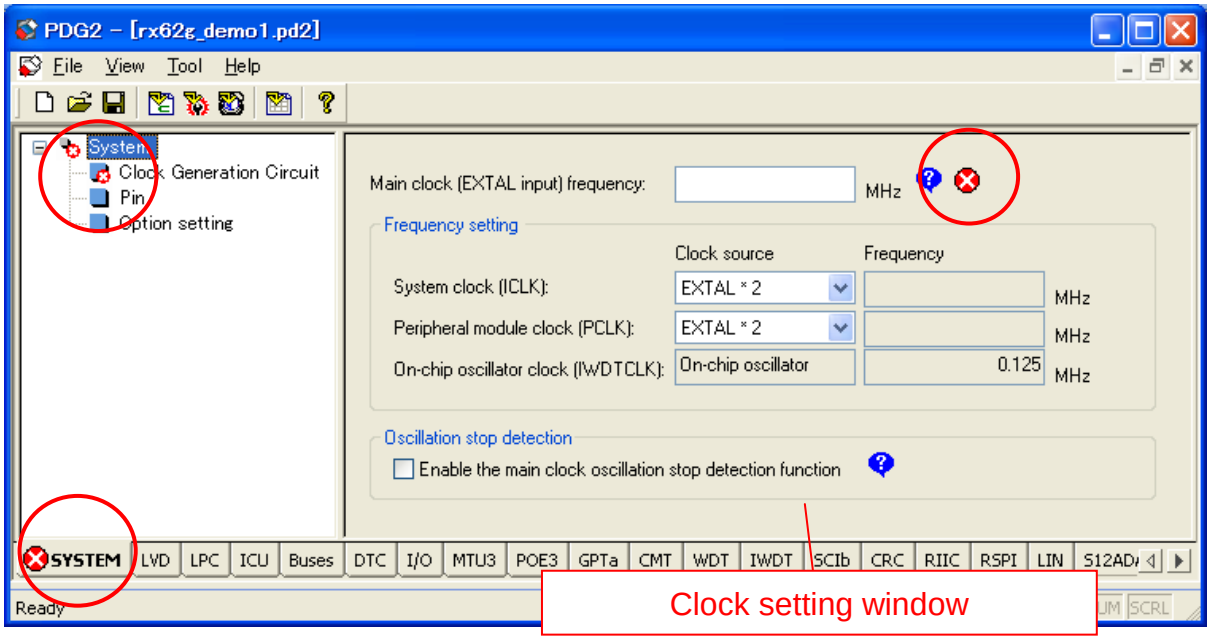
   Series : RX600
   Group : RX62G
   Type : R5F562GAADFP

   Note: If another type of chip is mounted on your RSK board, select corresponding CPU type.

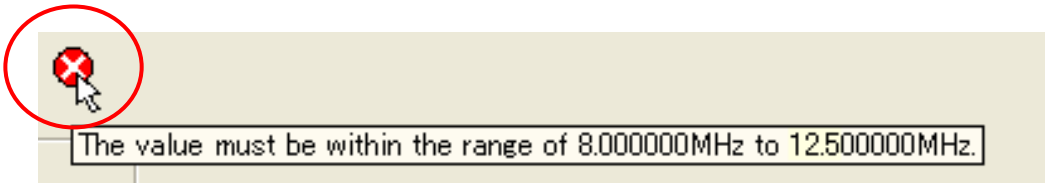


(2) Initial state **PDG**

-The clock setting window opens and the error icons are displayed in the initial state.



Place the mouse pointer on the error icon, then the contents of error is displayed.



There are 3 types of icons in Peripheral Driver Generator

Error
The setting is not allowed.
The source filese cannot be generated if there is an error setting.

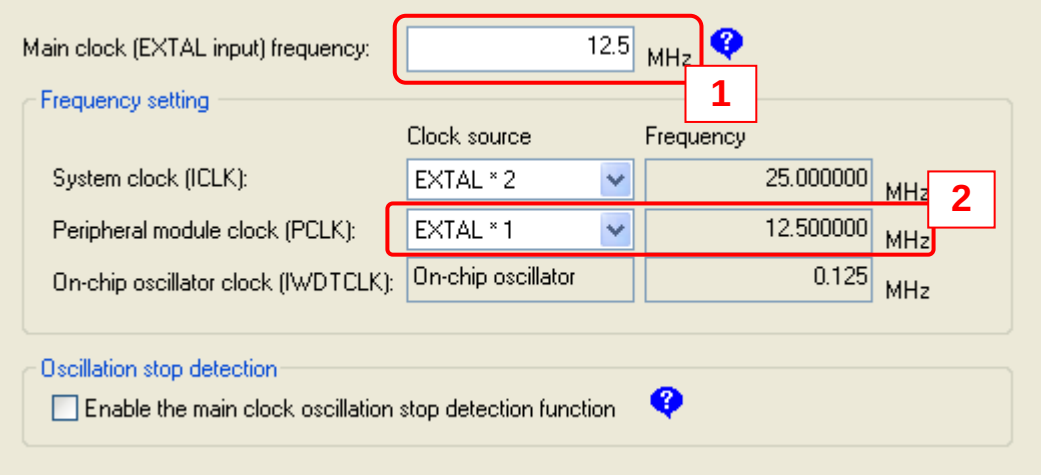
Warning
The setting is possible but may be wrong.
Source files can be generated.

Information
Additional information for the complex setting.

Only icons on the setting window can display the tooltip.

(3) Clock setting **PDG**

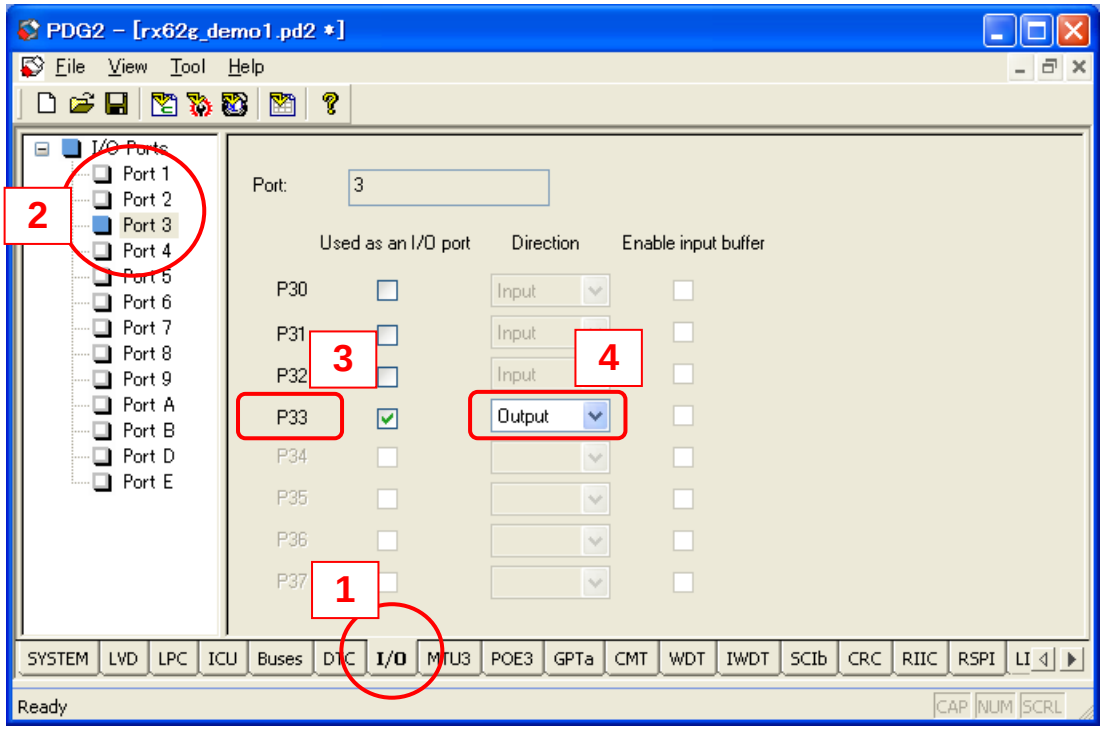
1. It is necessary to set the main (EXTAL) clock frequency first.
   External clock frequency of the RSK board is 12.5 MHz. Input “12.5” into the edit box.
2. PCLK is used in 12.5 MHz.
   Select the multiplication "EXTAL x 1" to set the PCLK to 12.5 MHz.



(4) I/O Port setting **PDG**

The LED3 on RSK is connected to P33 so set P33 to output port.

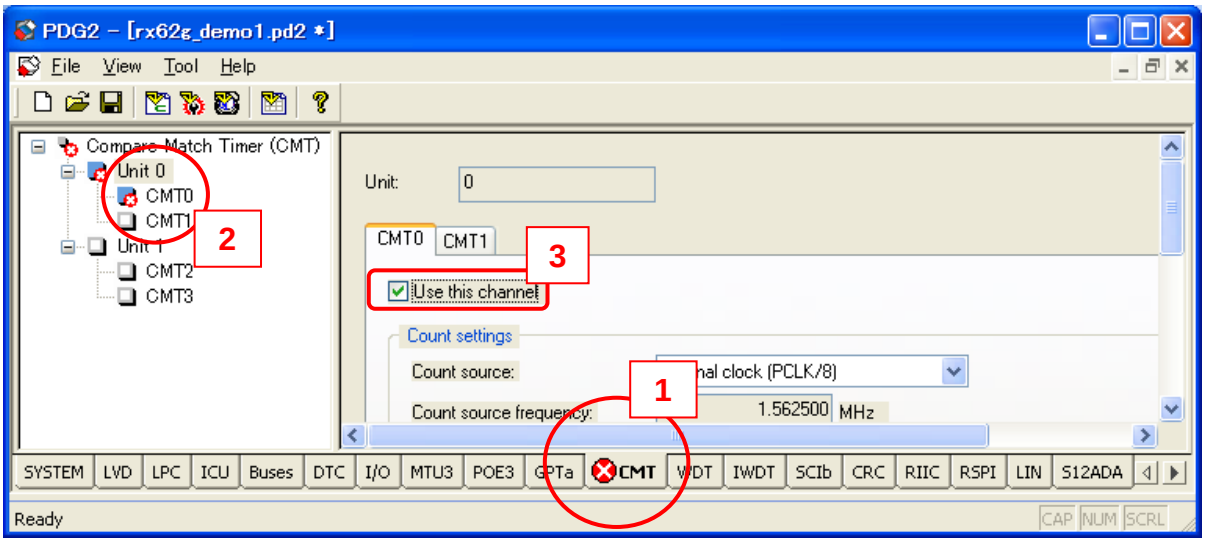
1. Select “I/O” tab
2. Select “Port 3”
3. Check “Pn3”
4. Select “Output”



(5) CMT setting-1 **PDG**

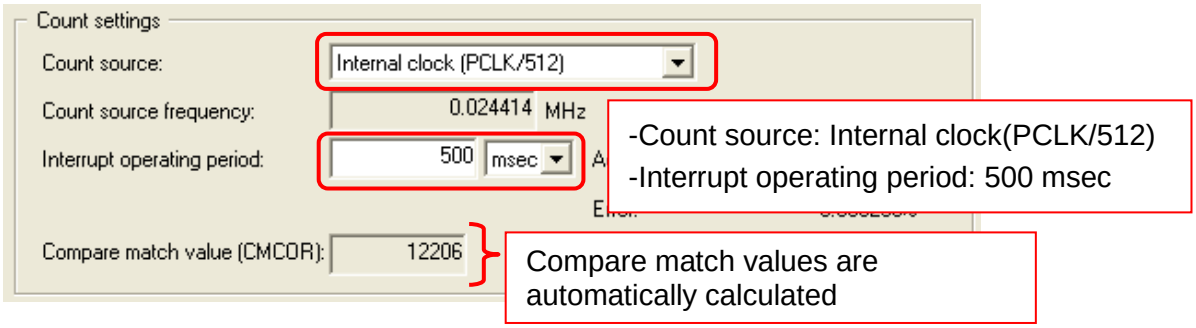
In this tutorial, CMT0 is used.

1. Select "CMT" tab
2. Select "CMT0"
3. Check "Use this channel"



(6) CMT setting-2 **PDG**

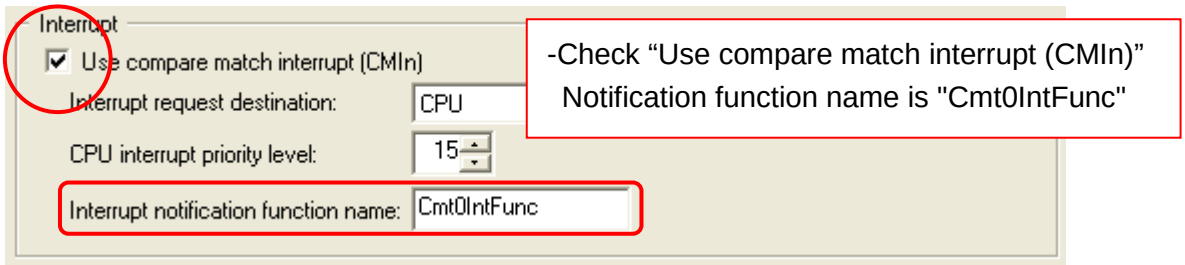
Set the other items as follows.



(7) CMT setting-3 **PDG**

Set the interrupt notification functions.

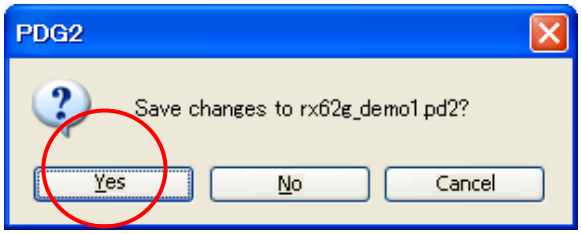
This functions are called when the interrupt occurs.



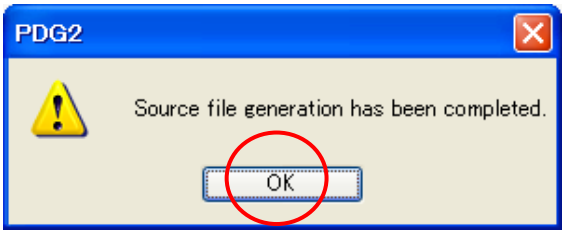
(8) Generating source files **PDG**

1. To generate source files, click  on the tool bar.

2. Save confirmation dialog box is displayed. Click [OK].

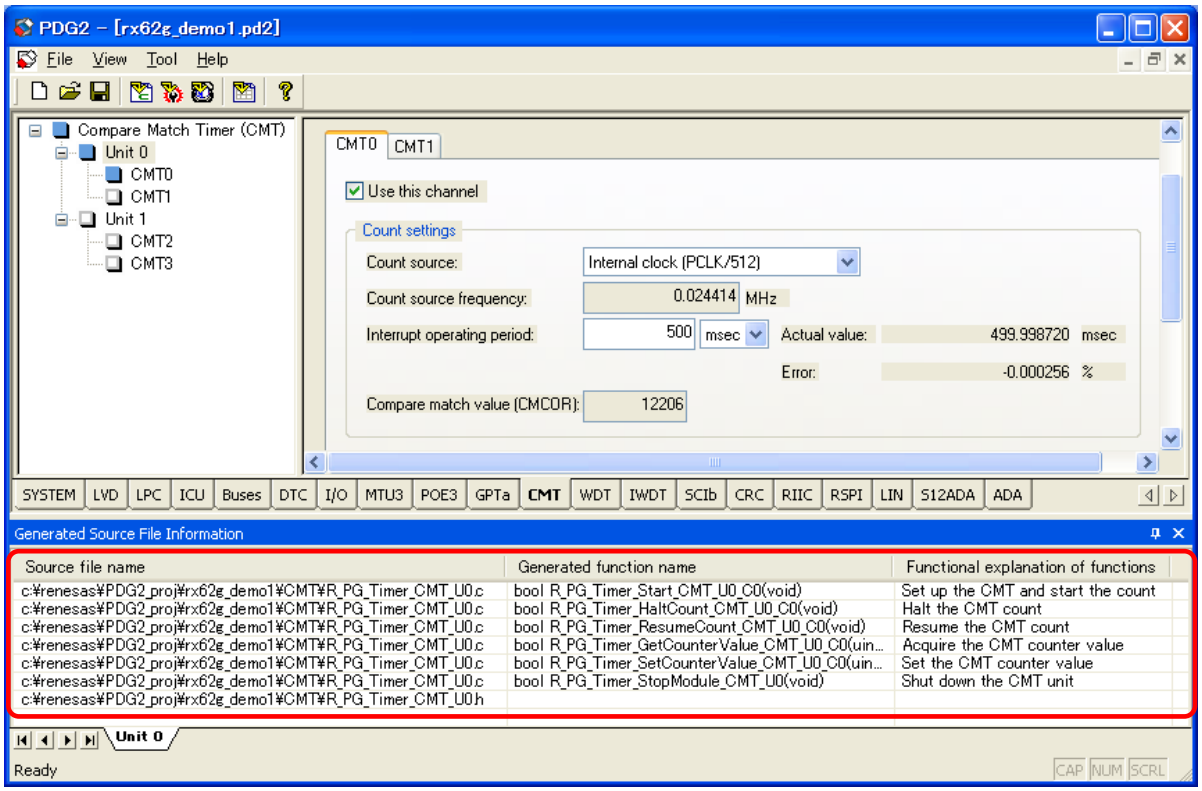


3. Click [OK] on the message box.



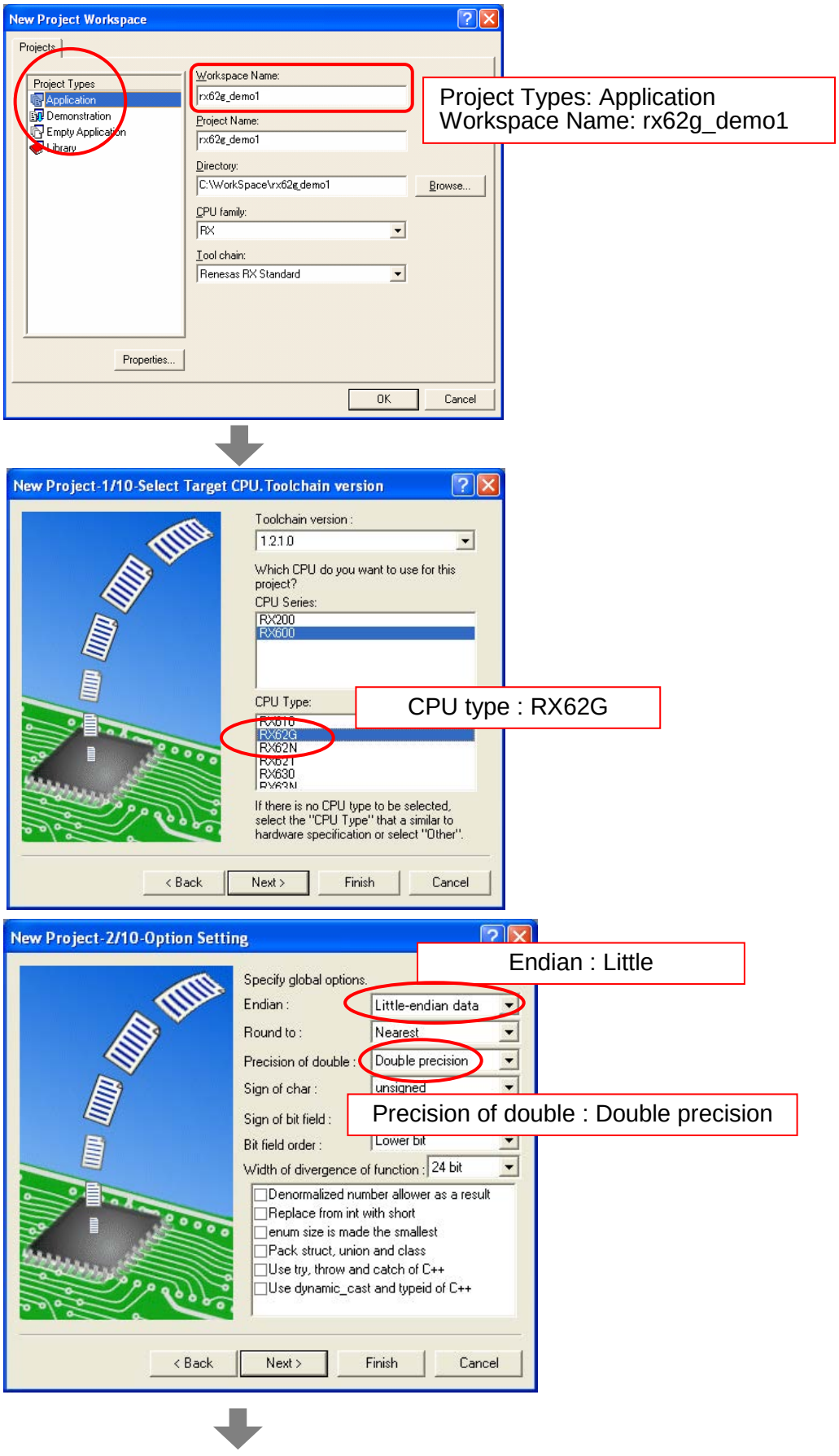
4. Generated functions are listed in lower panel.

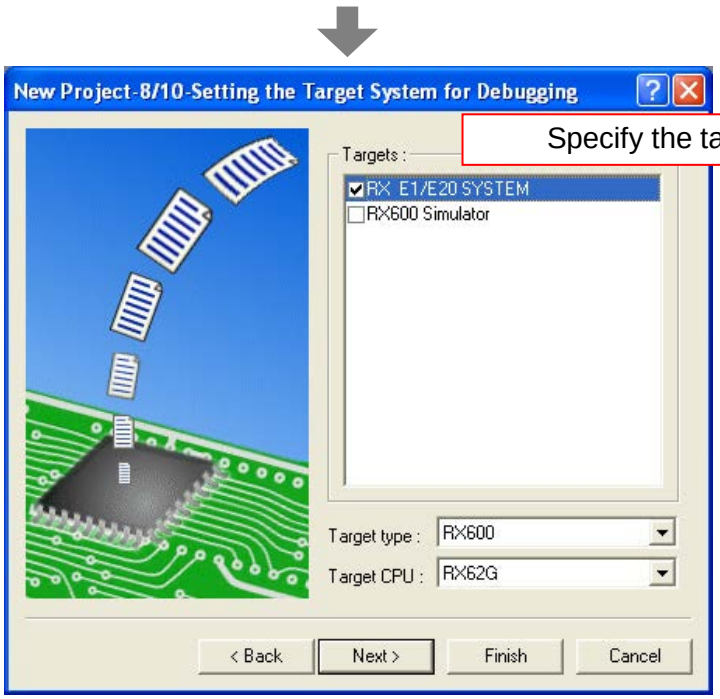
   By double clicking the line of function, source file can be opened.

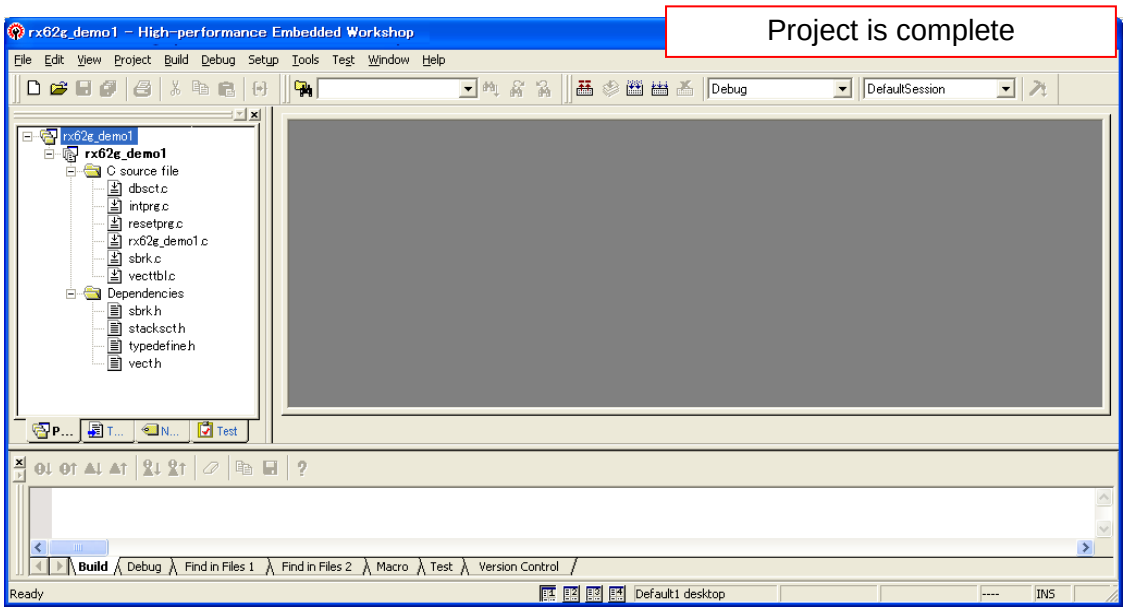


(9) Preparing the High-performance Embedded Workshop project **HEW**

Start the High-performance Embedded Workshop and make RX62G workspace.




New Project-8/10-Setting the Target System for Debugging
Specify the target emulator.
Targets :
RX E1/E20 SYSTEM
RX600 Simulator
Target type : RX600
Target CPU : RX62G
< Back
Next >
Finish
Cancel

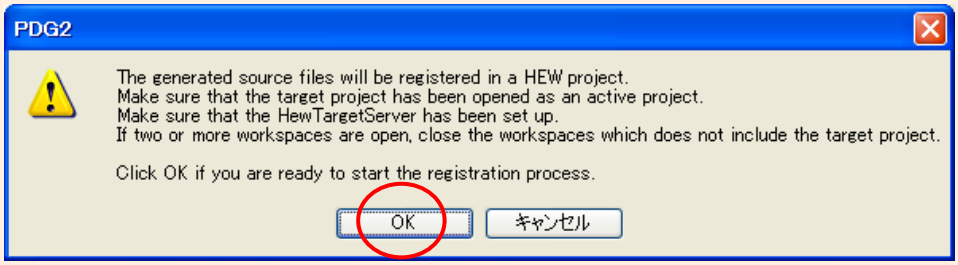

Project is complete
rx62g_demo1 - High-performance Embedded Workshop
File Edit View Project Build Debug Setup Tools Test Window Help
Debug
DefaultSession
rx62g_demo1
C source file
dbsct.c
intprg.c
resetprg.c
rx62g_demo1.c
sbrk.c
vecttbl.c
Dependencies
sbrk.h
stacksct.h
typedefine.h
vect.h
Build Debug Find in Files 1 Find in Files 2 Macro Test Version Control
Ready
Default1 desktop
INS

(10) Adding the generated source files to the High-performance Embedded Workshop project

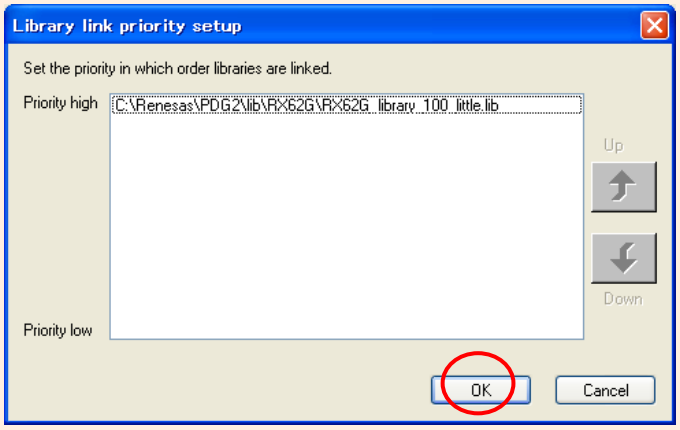
PDG

1. To add source files to High-performance Embedded Workshop, click  on the tool bar.

2. Click [OK] on the confirmation dialog box.

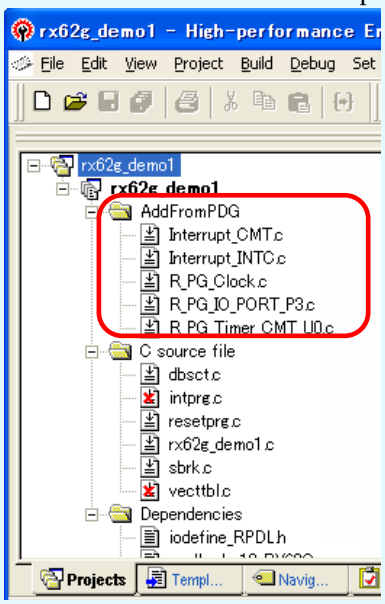


3. This is a linkage setting of Renesas Peripheral Driver Library.
When using multiple lib files, linkage order can be set in this dialog box.



HEW

4. Source files are added to High-performance Embedded Workshop.
Added source files are put in "AddFromPDG" folder.



Source files are registered via HEW Target Server.
Make sure that the HEW Target Server has been set up before executing registration.
For details, refer Peripheral Driver Generator user's manual.

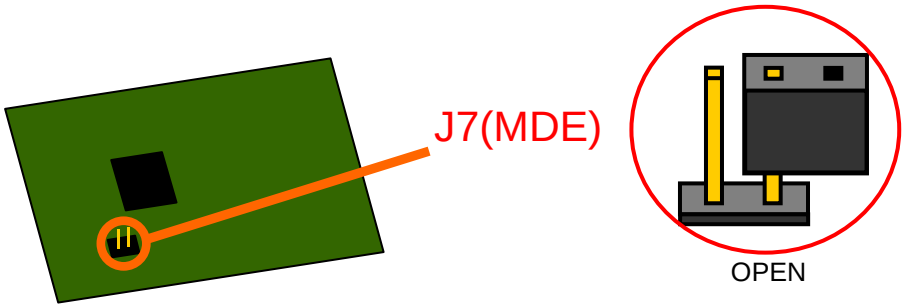
(11) Making the program on High-performance Embedded Workshop **HEW**

By changing the part of “main” function, make the following program on High-performance Embedded Workshop.

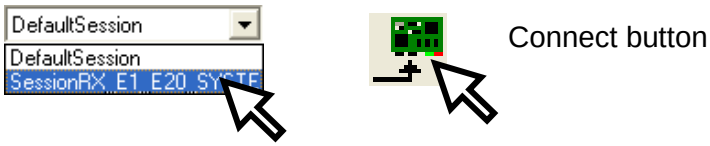
```
//Include "R_PG_< project name>.h"
#include "R_PG_rx62g_demo1.h"

bool led=false;

void main(void)
{
	//Set up the clock
	R_PG_Clock_Set();
	//Set up port P33
	R_PG_IO_PORT_Write_P33(1); // Initial output value
	R_PG_IO_PORT_Set_P33();
	//Set up CMT0 and start count
	R_PG_Timer_Start_CMT_U0_C0();
	while(1);
}
// Compare match interrupt notification function
void Cmt0IntFunc(void)
{
	if( led ){
		//Turn off the LED
		R_PG_IO_PORT_Write_P33(1);
		led = false;
	}
	else{
		//Turn on the LED
		R_PG_IO_PORT_Write_P33(0);
		led = true;
	}
}
```

(12) Connecting to the emulator, building the program and executing **HEW**

1. Before connecting the emulator, make sure that J7(MDE) on RSK board is “OPEN” to set CPU to little endian.



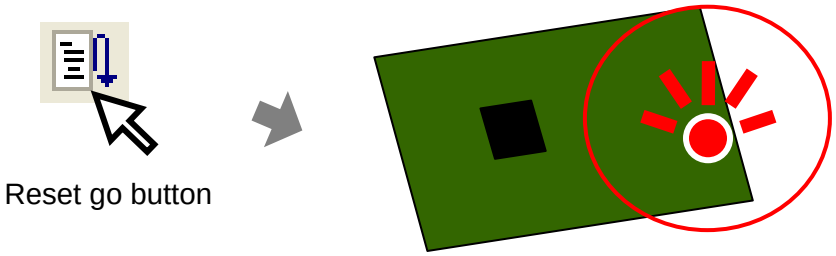
2. Connect to the emulator



3. Just by clicking [Build] button, program can be built because Renesas Peripheral Driver Library and include directory are automatically registered in build setting.

4. Download the program

5. Execute the program and see the LED on RSK board.

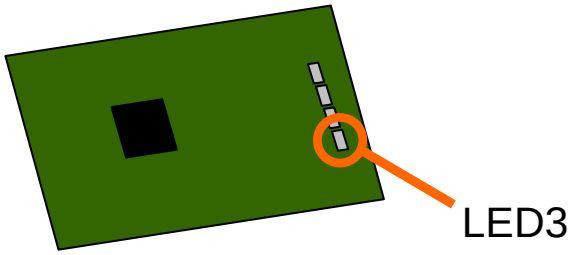


4.2 An LED blinking on the PWM output of the multi-function timer pulse unit 3 (MTU3)

The LED3 on RSK board is connected to P33. This port can also be used as PWM output pin (MTIOC3A) of the multi-function timer pulse unit 3. In this tutorial, the multi-function timer pulse unit 3 will be set up to operate in PWM mode 1 and the PWM output will blink the LED3 as follows.

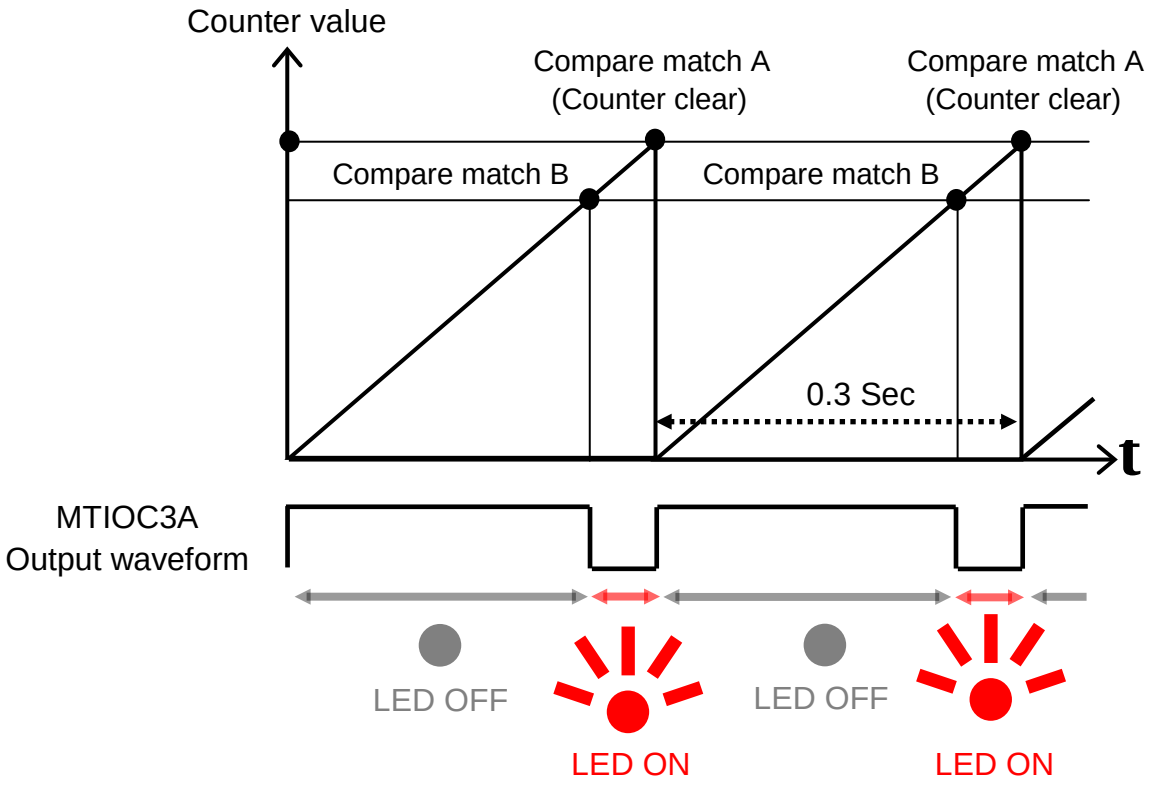
Note : If there is a switch that enables/disables P33(MTIOC3A) on the RSK board, enable it.

The LED3 turns on when the output from P33 is 0, and turns off when the output is 1.



The MTU3 channel 3 (MTU3) will be operated in PWM mode 1. In PWM mode 1, the output signal is controlled by compare match A and B.

Operation of the timer to be set
- Output 0 at compare match B -> LED turns on
- Output 1 at compare match A -> LED turns off
- Clear the counter at compare match A (Intervals of 0.3 sec)



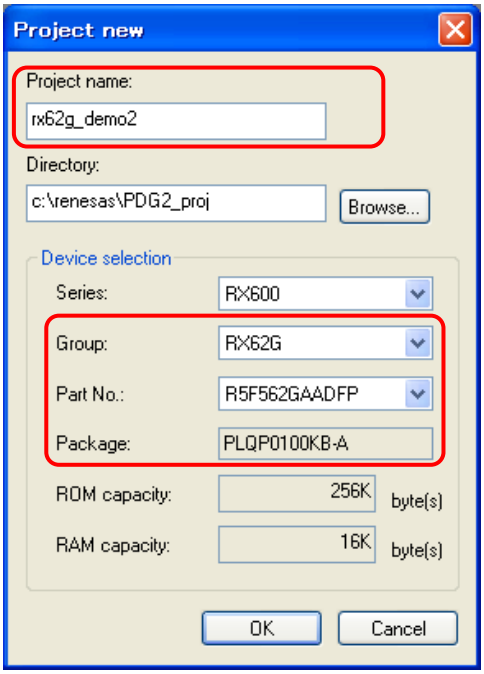
PDG

(1) Making the Peripheral Driver Generator project

Make the new Peripheral Driver Generator project “rx62g_demo2”. For details on how to make the new Peripheral Driver Generator project, refer to section 4.1(1), Making the Peripheral Driver Generator project. Set the CPU type as follows.

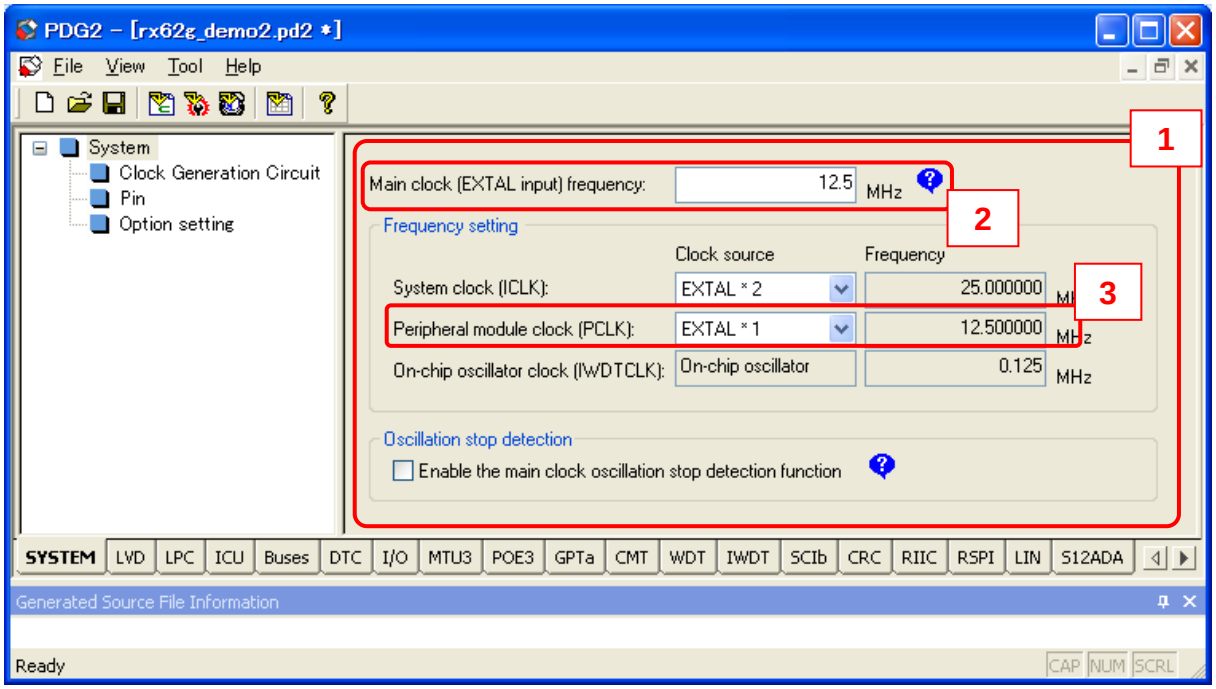
Series : RX600
Group : RX62G
Type : R5F562GAADFP

Note: If another type of chip is mounted on your RSK board, select corresponding CPU type.



(2) Clock setting PDG

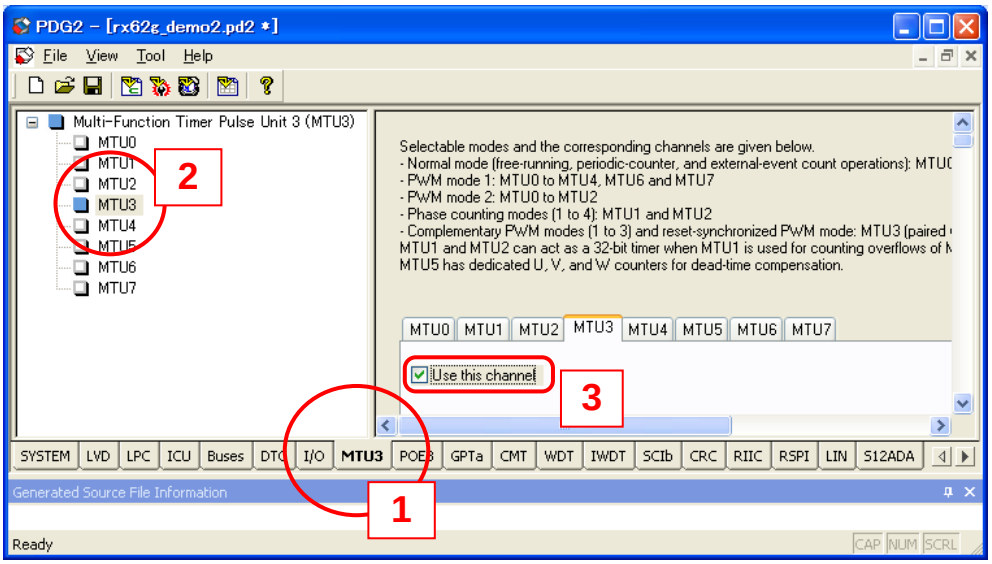
1. The clock setting window opens and the error icons are displayed in the initial state. For icons such as and displayed on window, refer to section 4.1(2), Initial state.
2. External clock frequency of the RSK board is 12.5 MHz. Input “12.5” into the edit box.
3. PCLK is used in 12.5 MHz. Select the multiplication "EXTAL * 1" to set the PCLK to 12.5 MHz.



(3) MTU3 setting-1 **PDG**

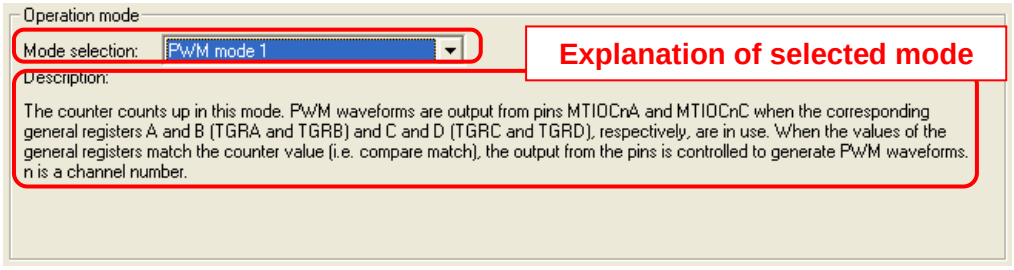
Opening MTU3 channel 3(MTU3) setting window

1. Select "MTU3" tab.
2. Select "MTU3" on tree view.
3. Check "Use this channel".



(4) MTU3 setting-2 **PDG**

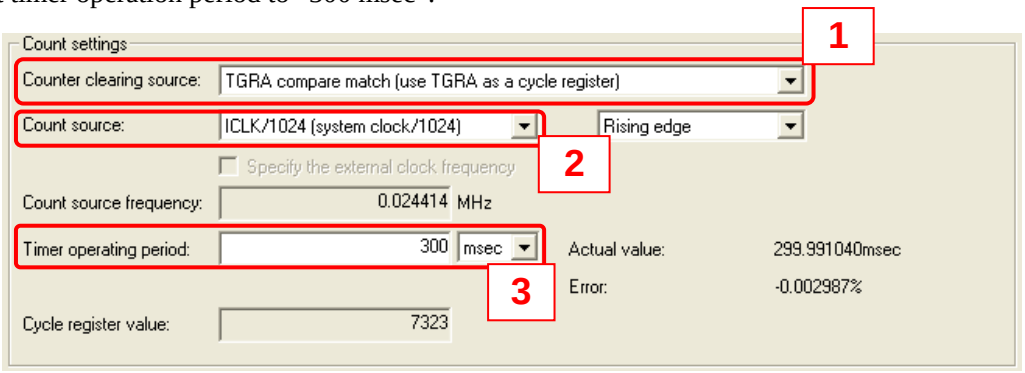
Select "PWM mode 1" for the operation mode.



(5) MTU3 setting-3 **PDG**

The counter setting is as follows.

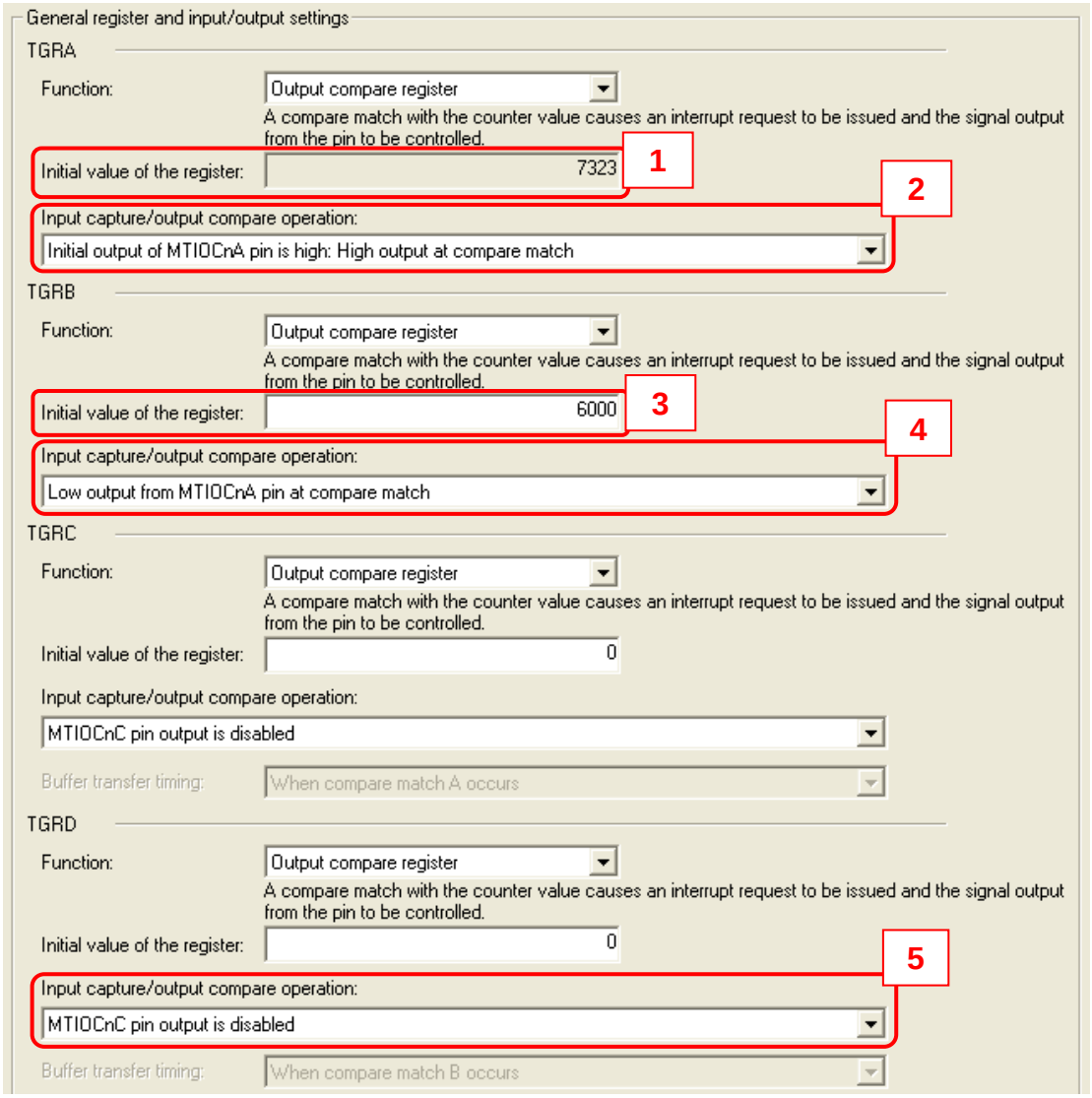
1. Select "TGRA compare match (use TGRA as a cycle register)" for a counter clearing source.
2. Select "ICLK/1024 (system clock/1024)" for a count source.
3. Set timer operation period to "300 msec".



(6) MTU3 setting-4 **PDG**

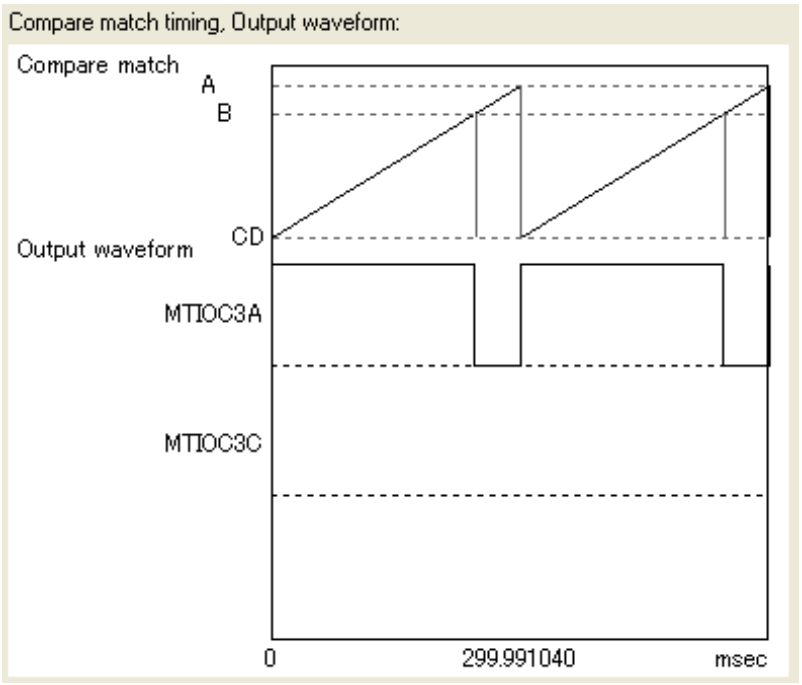
General register setting is as follows.

1. The TGRA is selected as a counter clearing source in the counter setting. Then the TGRA value is calculated from the count source frequency and the timer operating period.
2. Select "Initial output of MTIOCnA pin is high: High output at compare match" for TGRA output compare operation.
3. Set TGRB initial value to "6000".
4. Select "Low output from MTIOCnA pin at compare match" for TGRB output compare operation.
5. The MTIOCnC output is not used in this tutorial. Select "MTIOCnC pin output is disabled" for TGRD output compare operation.



(7) MTU3 setting-5 PDG

The compare match timing and the output waveform are displayed in a diagram.



(8) Generating source files PDG

To generate source files, click  on the tool bar. For details on generating source files, refer to section 4.1(8), Generating source files.

(9) Preparing the High-performance Embedded Workshop project HEW

Start the High-performance Embedded Workshop and make RX62G workspace. For details on making High-performance Embedded Workshop project, refer to section 4.1(9), Preparing the High-performance Embedded Workshop project.

PDG

(10) Adding the generated source files to the High-performance Embedded Workshop project

To add the generated source files to High-performance Embedded Workshop, click  on the tool bar. For details on adding the source files to High-performance Embedded Workshop project, refer to section 4.1(10), Adding the generated source files to the High-performance Embedded Workshop project.

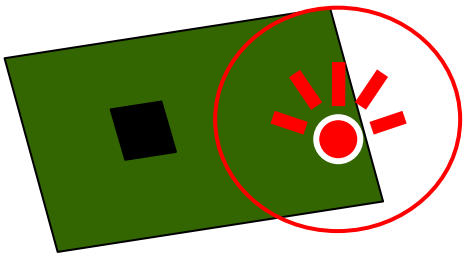
(11) Making the program on High-performance Embedded Workshop **HEW**

By changing the part of “main” function, make the following program on High-performance Embedded Workshop.

```
//Include "R_PG_<project name>.h"
#include "R_PG_rx62g_demo2.h"

void main(void)
{
	//Set up the clock
	R_PG_Clock_Set();

	//Set up MTU3 Channel 3
	R_PG_Timer_Set_MTU_U0_C3();

	//Start the count of MTU3 Channel 3
	R_PG_Timer_StartCount_MTU_U0_C3();

	while(1);
}
```

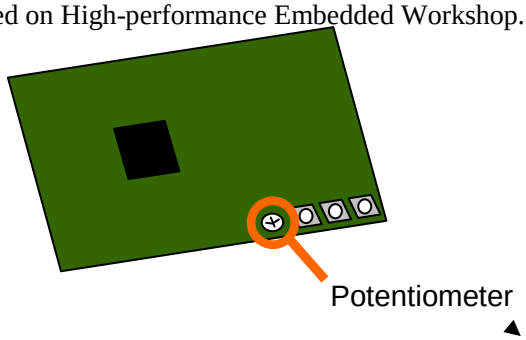
(12) Connecting to the emulator, building the program and executing **HEW**

Execute the program and see the LED blinking on RSK board. For details on connecting to the emulator, building the program, and executing the program, refer to section 4.1(12), connecting to the emulator, building the program and executing.



4.3 Continuously scanning on10-Bit A/D converter (ADA)

In RX62G RSK board, the potentiometer is connected to AN0 analog input. In this tutorial, the 10-Bit A/D converter (ADA) will be set up to execute A/D conversion continuously. And the result of A/D conversion will be monitored on High-performance Embedded Workshop.



Note : If there is a switch that enables/disables P05 on the RSK board, enable it.

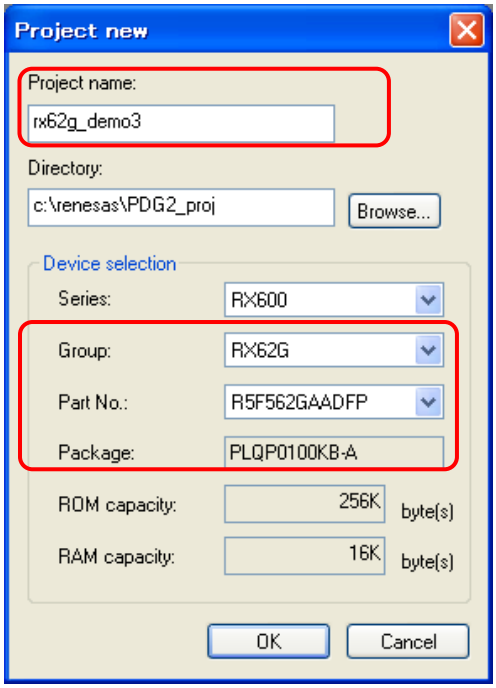
(1) Making the Peripheral Driver Generator project

PDG

Make the new Peripheral Driver Generator project “rx62g_demo3”. For details on how to make the new Peripheral Driver Generator project, refer to section 4.1(1), Making the Peripheral Driver Generator project. Set the CPU type as follows.

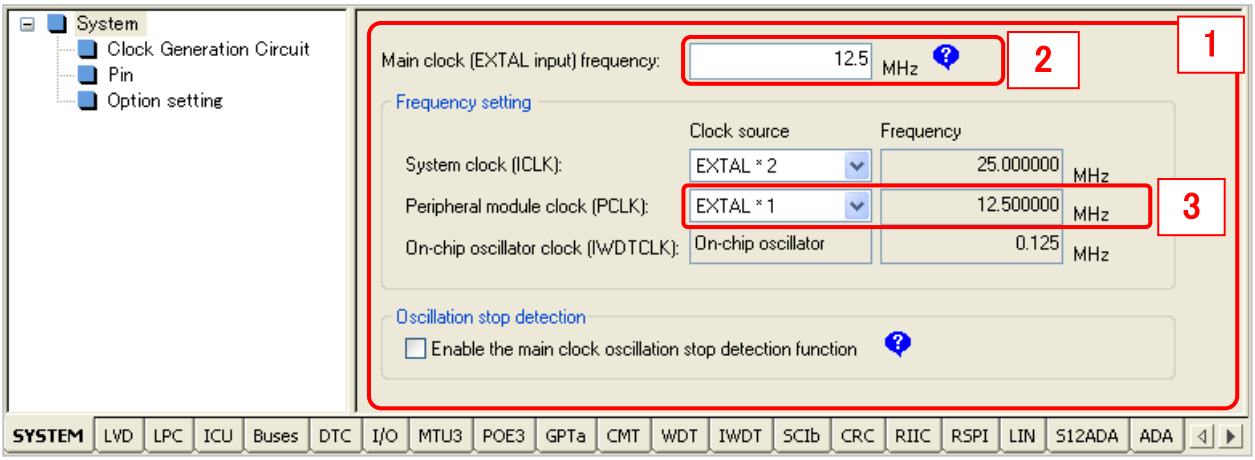
Series : RX600
Group : RX62G
Type : R5F562GAADFP

Note: If another type of chip is mounted on your RSK board, select corresponding CPU type.



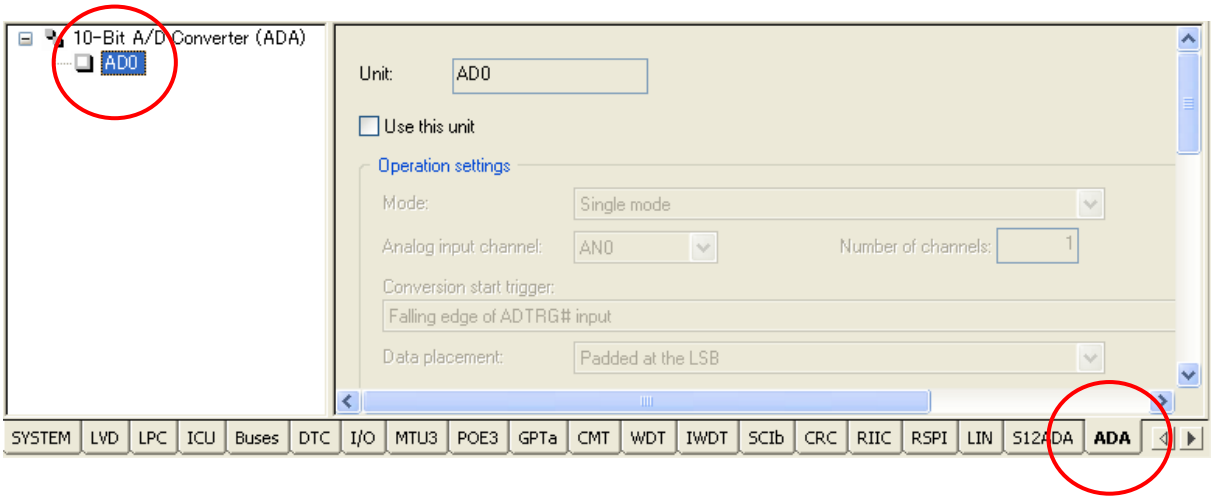
(2) Clock setting **PDG**

1. The clock setting window opens and the error icons are displayed in the initial state. For icons such as and displayed on window, refer to section 4.1(2), Initial state.
2. External clock frequency of the RSK board is 12.5 MHz. Input “12.5” into the edit box.
3. PCLK is used in 12.5 MHz. Select the multiplication "EXTAL x 1" to set the PCLK to 12.5 MHz



(3) A/D converter setting-1 **PDG**

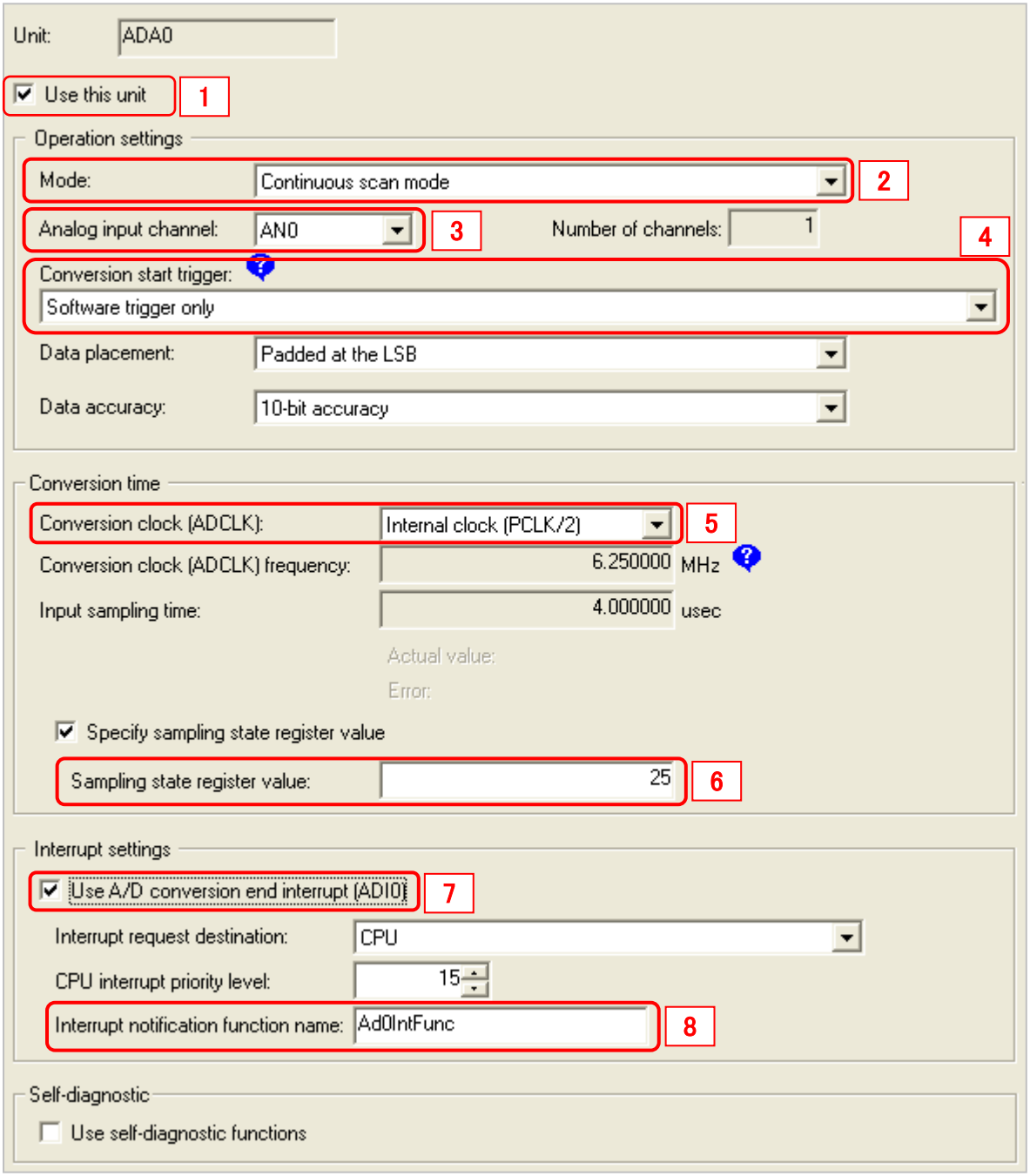
Select “ADA” tab and click ADA0 on tree view.



(4) A/D converter setting-2 **PDG**

Make the following setting for ADA0.

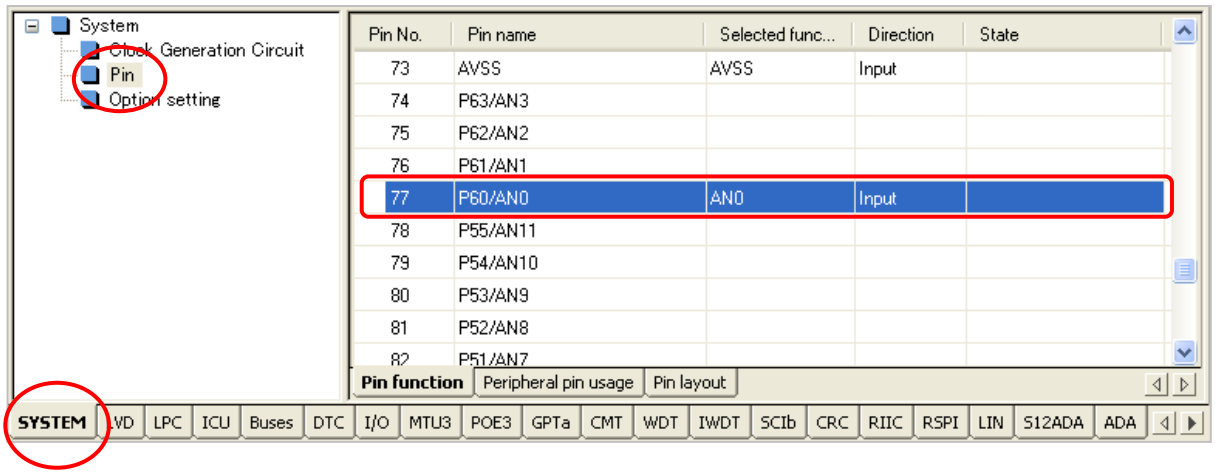
1. Check "Use this unit".
2. Select "Continuous scan mode" for the operation mode.
3. Select "AN0" for the analog input channel.
4. Select "Software trigger only" for the conversion start trigger.
5. Select "Internal clock (PCLK/2)" for the conversion clock.
6. Set the sampling state register value to 25.
7. Check "Use A/D conversion end interrupt (ADI0)".
8. Set A/D conversion end interrupt notification function name to "Ad0IntFunc".



(5) Checking the pin usage **PDG**

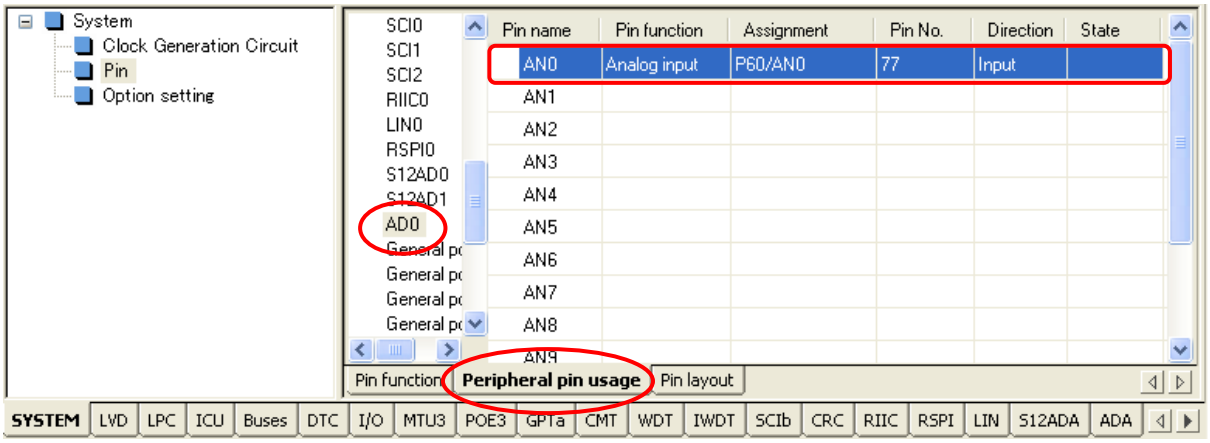
- It is possible to check the usage of pins on the pin function windows

1. After setting up the ADA0, select “SYSTEM” tab and click “Pin” on the tree view.
2. On the Pin function window, you can see that No.77 pin is used as AN0.



- State of pin usage for each peripheral module is displayed in the Peripheral Pin Usage Window

Select Peripheral pin usage sheet and click ADA0 to check the usage of AN0 pin.



(6) Generating source files PDG

To generate source files, click  on the tool bar. For details on generating source files, refer to section 4.1(8), Generating source files.

(7) Preparing the High-performance Embedded Workshop project HEW

Start the High-performance Embedded Workshop and make RX62G workspace. For details on making High-performance Embedded Workshop project, refer to section 4.1(9), Preparing the High-performance Embedded Workshop project.

(8) Adding the generated source files to the High-performance Embedded Workshop project PDG

To add the generated source files to High-performance Embedded Workshop, click  on the tool bar. For details on adding the source files to High-performance Embedded Workshop project, refer to section 4.1(10), Adding the generated source files to the High-performance Embedded Workshop project.

(9) Making the program on High-performance Embedded Workshop HEW

By changing the part of “main” function, make the following program on High-performance Embedded Workshop.

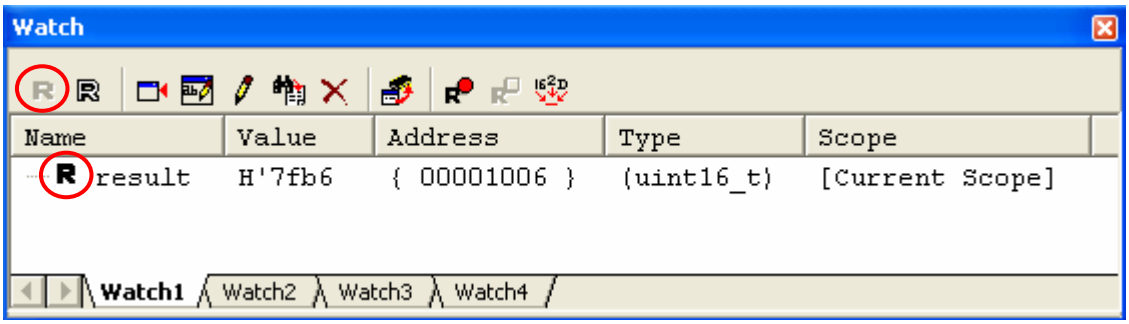
```
//Include "R_PG_<project name>.h"
#include "R_PG_rx62g_demo3.h"

void main(void)
{
    //Set up the clock
    R_PG_Clock_Set();

    //Set up ADA0
    R_PG_ADC_10_Set_AD0();

    //Start A/D conversion
    R_PG_ADC_10_StartConversionSW_AD0();

    while(1);
}

// Variable to store the result
uint16_t result;

// A/D conversion end interrupt notification function
void Ad0IntFunc(void)
{
    // Get the result of conversion
    R_PG_ADC_10_GetResult_AD0(&result);
}
```

(10) Connecting to the emulator, building the program and downloading **HEW**

Buid the program and download it. For details on connecting to the emulator, building the program, and downloading refer to section 4.1(12), connecting to the emulator, building the program and executing.

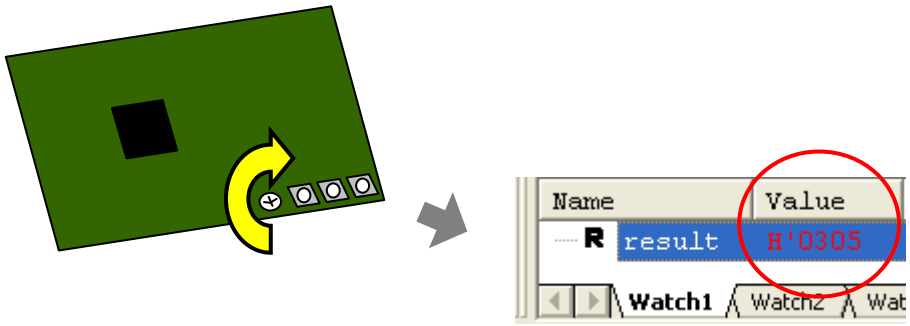
(11) Adding the variable of A/D conversion result to the watch window **HEW**

Open the Watch window and add the variable "result". Set "result" to the real time update to monitor the variable change during execution.



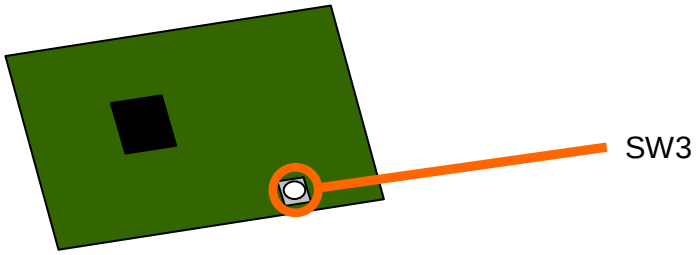
(12) Executing the program and monitoring the A/D conversion result **HEW**

Start the execution and screw the potentiometer to change the analog input voltage. The value of “result” on the watch window will change.



4.4 Triggering DTC by IRQ

In RX62G RSK board, switch 3 (SW3) is connected to IRQ3. In this tutorial, the data transfer controller (DTC) and IRQ3 will be set up and DTC transfer triggered by IRQ3 will be performed.



Note : If there is a switch that enables/disables IRQ3 on the RSK board, enable it.

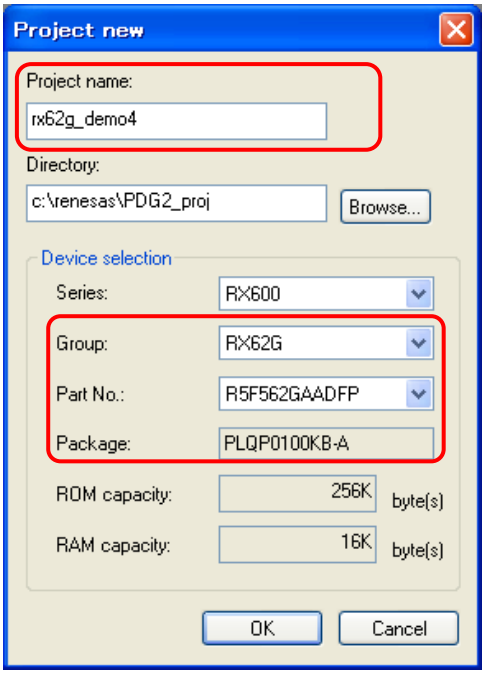
(1) Making the Peripheral Driver Generator project

Make the new Peripheral Driver Generator project “rx62g_demo4”. For details on how to make the new Peripheral Driver Generator project, refer to section 4.1(1), Making the Peripheral Driver Generator project. Set the CPU type as follows.

Series : RX600
Group : RX62G
Type : R5F562GAADFP

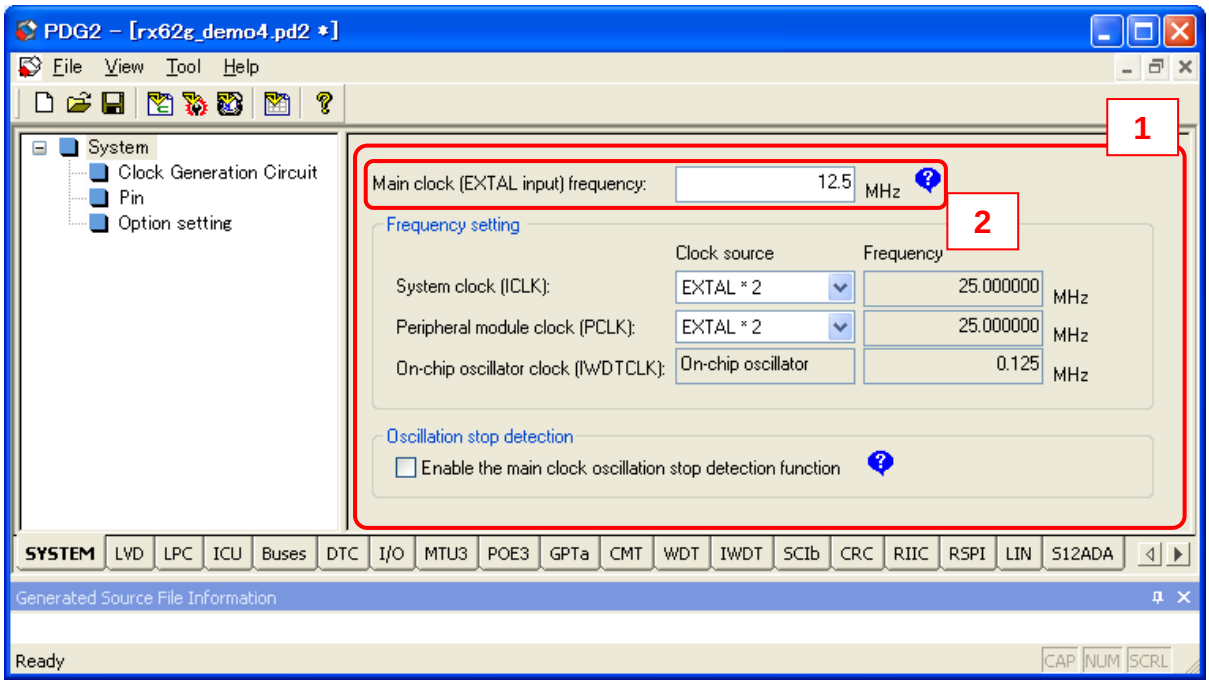
Note:
If another type of chip is mounted on your RSK board, select corresponding CPU type.
Only R5F562GAxxxx is targeted in this tutorial.

PDG



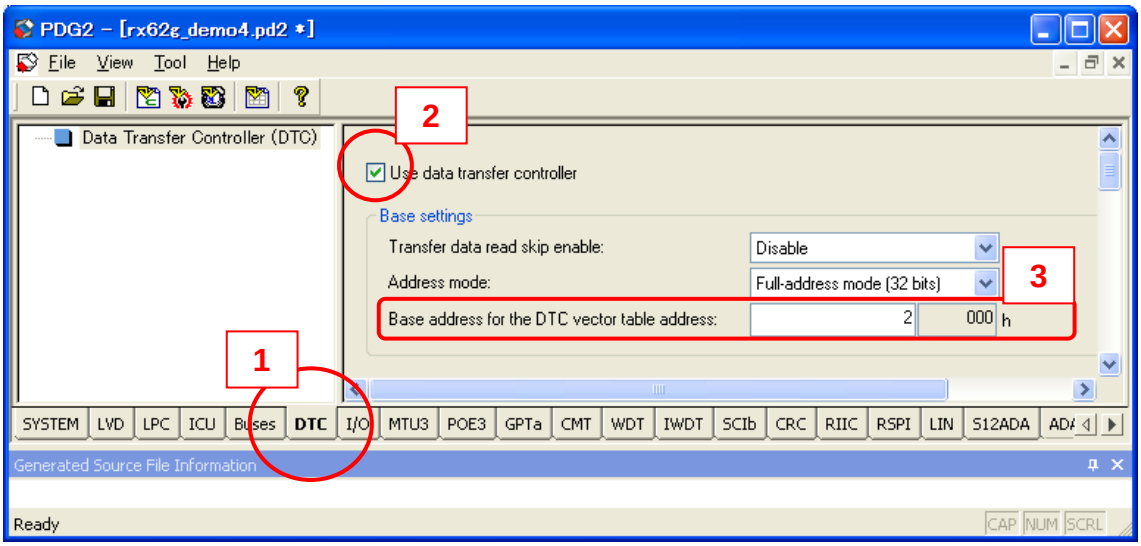
(2) Clock setting **PDG**

1. The clock setting window opens and the error icons are displayed in the initial state. For icons such as and displayed on window, refer to section 4.1(2), Initial state.
2. External clock frequency of the RSK board is 12.5 MHz. Input “12.5” into the edit box.



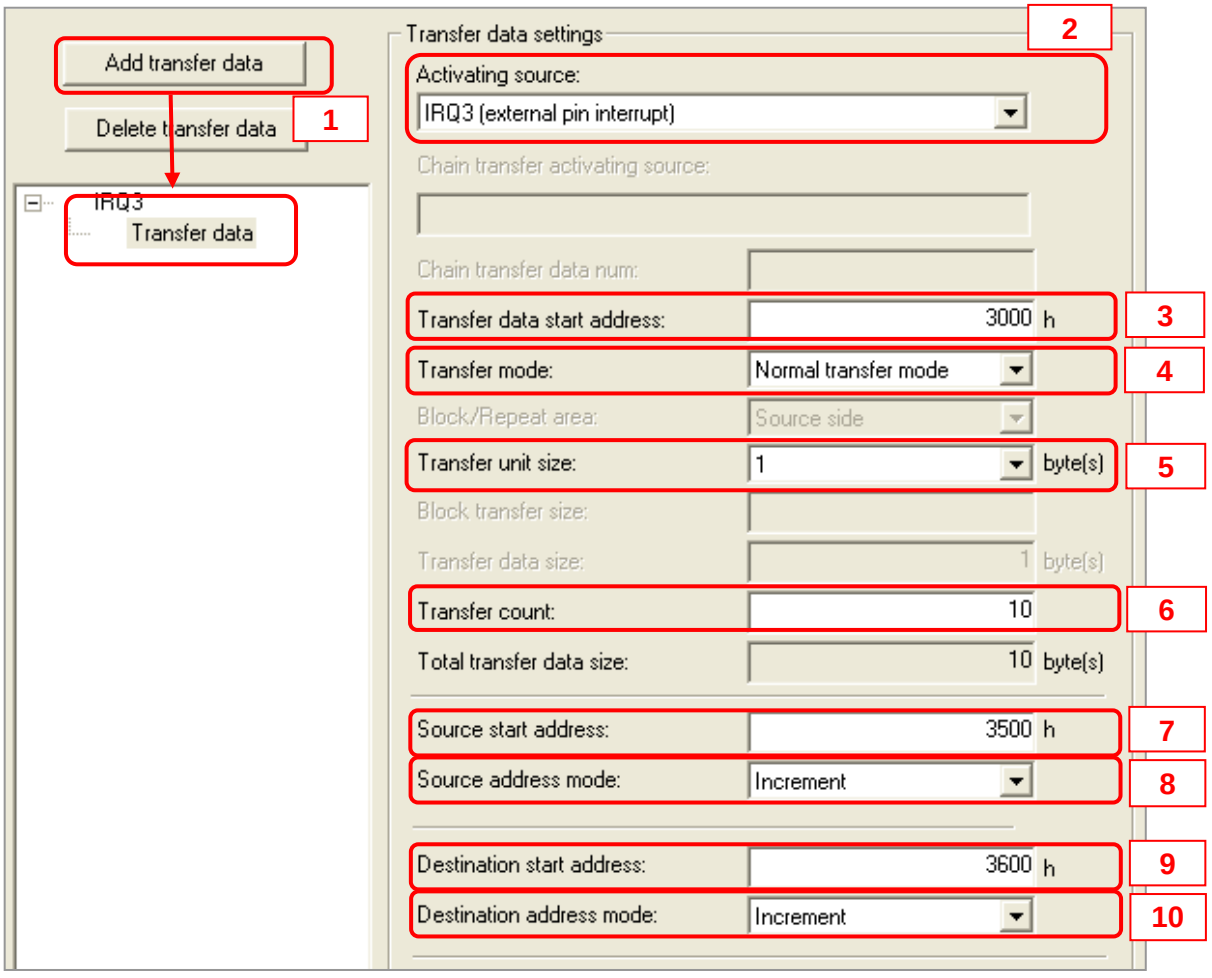
(3) DTC setting-1 **PDG**

1. Select “DTC” tab to open the DTC setting window.
2. Check "Use data transfer controller".
3. The DTC vector table will be allocated from 2,000. Input “2” into the edit box.



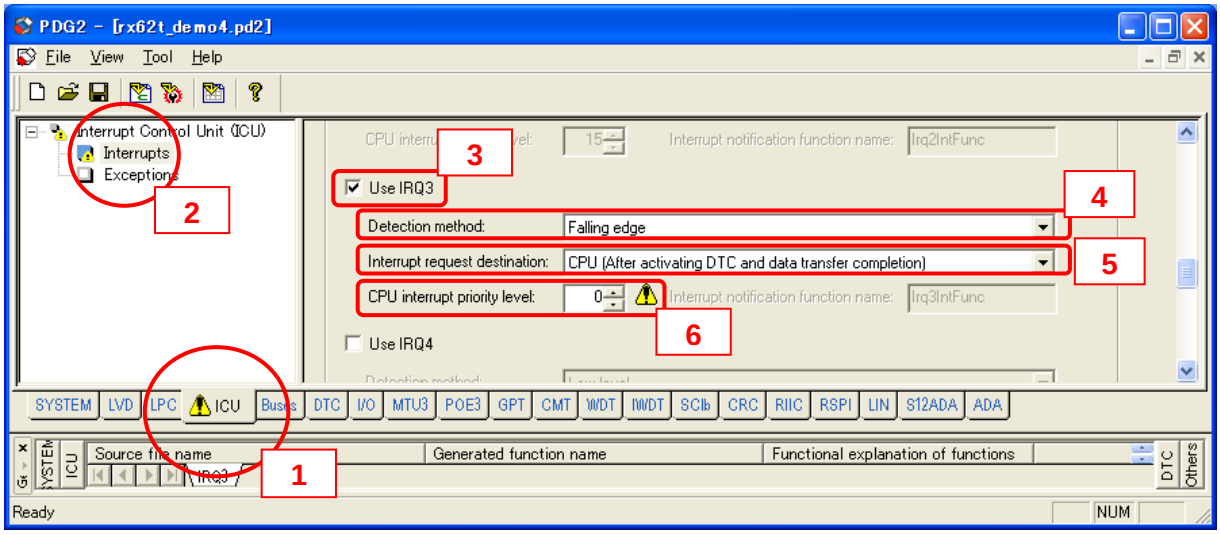
(4) DTC setting-2 **PDG**

1. Click [Add transfer data] to add the transfer data.
2. Select “IRQ3 (external pin interrupt)” for the activating source.
3. Set the transfer data start address to “3000”.
4. Select “Normal transfer mode” for the transfer mode.
5. Select “1” for the transfer unit size.
6. Set transfer count to “10”.
7. Set the transfer source start address to “3500”.
8. Select “Increment” for the transfer source address mode.
9. Set the transfer destination start address to “3600”.
10. Select “Increment” for the transfer destination address mode.



(5) IRQ setting **PDG**

1. Select “ICU” tab to open the ICU setting window.
2. Click “Interrupts” on the tree view.
3. Check "Use IRQ3".
4. Select “Falling edge” for the detection method of IRQ3.
5. Select “CPU (After activating DTC and data transfer completion)”.
6. CPU interrupt will not be used then set the CPU interrupt priority level to “0”.



(6) Generating source files **PDG**

To generate source files, click  on the tool bar. For details on generating source files, refer to section 4.1(8), Generating source files.

(7) Preparing the High-performance Embedded Workshop project **HEW**

Start the High-performance Embedded Workshop and make RX62G workspace. For details on making High-performance Embedded Workshop project, refer to section 4.1(9), Preparing the High-performance Embedded Workshop project.

PDG

(8) Adding the generated source files to the High-performance Embedded Workshop project

To add the generated source files to High-performance Embedded Workshop, click  on the tool bar. For details on adding the source files to High-performance Embedded Workshop project, refer to section 4.1(10), Adding the generated source files to the High-performance Embedded Workshop project.

(9) Making the program on High-performance Embedded Workshop **HEW**

By changing the part of “main” function, make the following program on High-performance Embedded Workshop.

```
//Include "R_PG_<project name>.h"
#include "R_PG_rx62g_demo4.h"

//DTC vector table
#pragma address dtc_vector_table = 0x00002000
uint32_t dtc_vector_table [256];

//DTC transfer data storage area (IRQ3)
#pragma address dtc_transfer_data_IRQ3 = 0x00003000
uint32_t dtc_transfer_data_IRQ3 [2];

//Transfer source
#pragma address dtc_src_data = 0x00003500
uint8_t dtc_src_data [10] = "ABCDEFGHIJ";

//Transfer destination
#pragma address dtc_dest_data = 0x00003600
uint8_t dtc_dest_data [10];

void main(void)
{
      //initialize transfer destination
      int i;
      for(i=0; i<10; i++ ){
            dtc_dest_data[i] = 0;
      }

      // Set up the clock
      R_PG_Clock_Set();

      // Set up the DTC (e.g. vector table address)
      R_PG_DTC_Set();

      // Set up the DTC (transfer data of IRQ3)
      R_PG_DTC_Set_IRQ3();

      // Set up IRQ3
      R_PG_ExtInterrupt_Set_IRQ3();

      // Make the DTC be ready to the trigger
      R_PG_DTC_Activate();
      while(1);
}
```

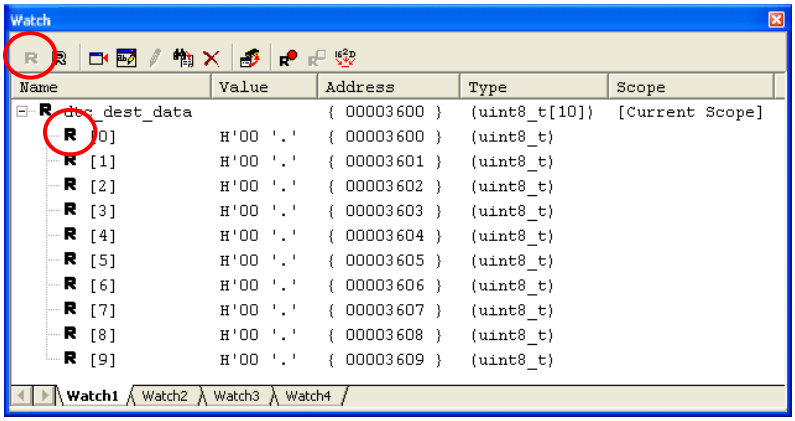
(10) Connecting to the emulator, building the program and downloading **HEW**

Build the program and download it. For details on connecting to the emulator, building the program, and downloading refer to section 4.1(12), connecting to the emulator, building the program and executing.

Note: When using RX Family C/C++ compiler package V.1.01 or later, the error message may be output in building the program. For details, refer to section 6.(5).

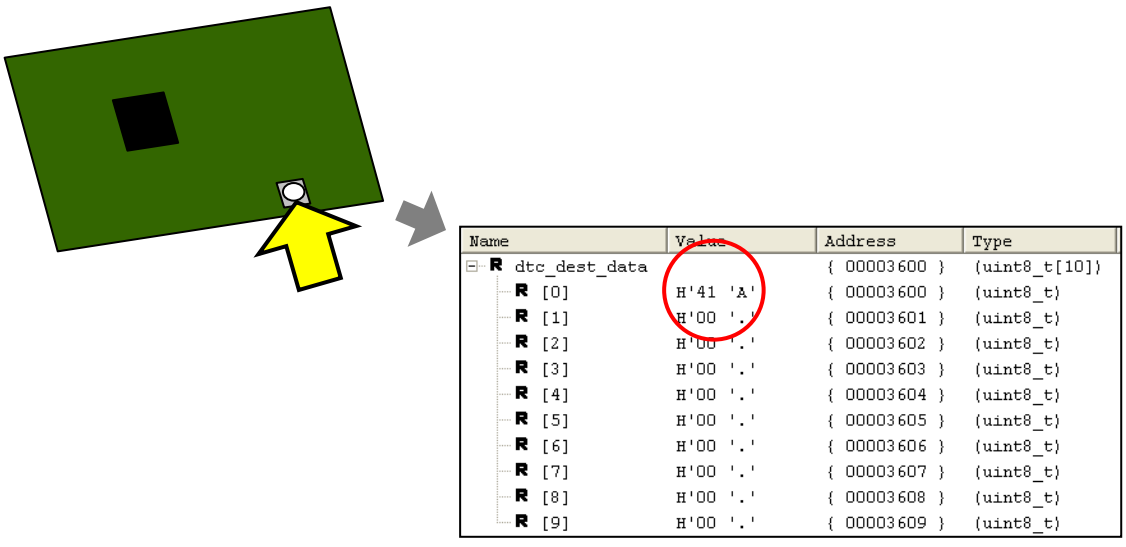
(11) Adding the variable of the transfer destination **HEW**

Open the Watch window and add the variable "dtc_dest_data". Expand the array and set it to the real time update to monitor the variable change during execution.



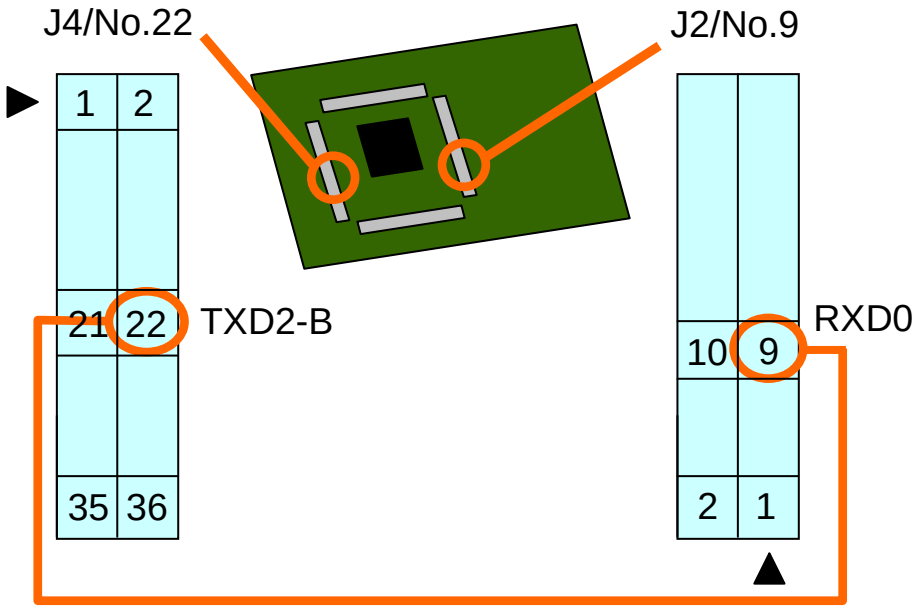
(12) Executing the program and monitoring the result of the transfer **HEW**

Start the execution and push the SW3. The value of “dtc_dest_data” on the watch window will change.



4.5 Data transfer between SCIb channels 0 and 2

In this tutorial, SCIb channel 0 and 2 will be set up to transfer data in asynchronous mode. Connect the transmission pin of channel 2 (TXD2-B) and the reception pin of channel 0 (RXD0) on the RSK board as follows.



Note : If there are switches that enables/disables TXD2-B and RXD0 on the RSK board, enable it.

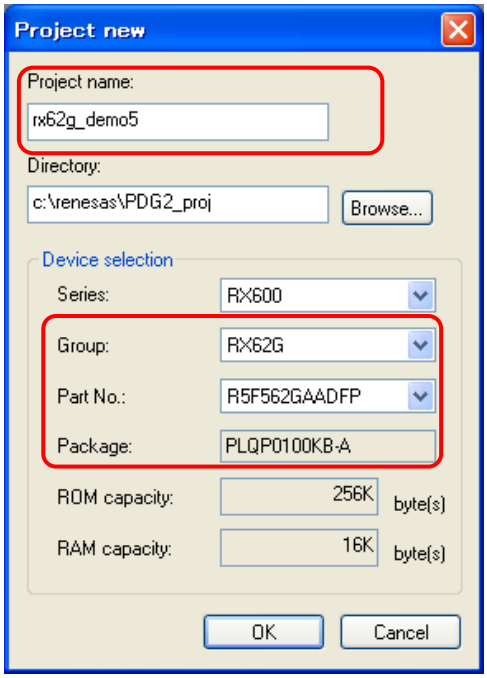
PDG

(1) Making the Peripheral Driver Generator project

Make the new Peripheral Driver Generator project “rx62g_demo5”. For details on how to make the new Peripheral Driver Generator project, refer to section 4.1(1), Making the Peripheral Driver Generator project. Set the CPU type as follows.

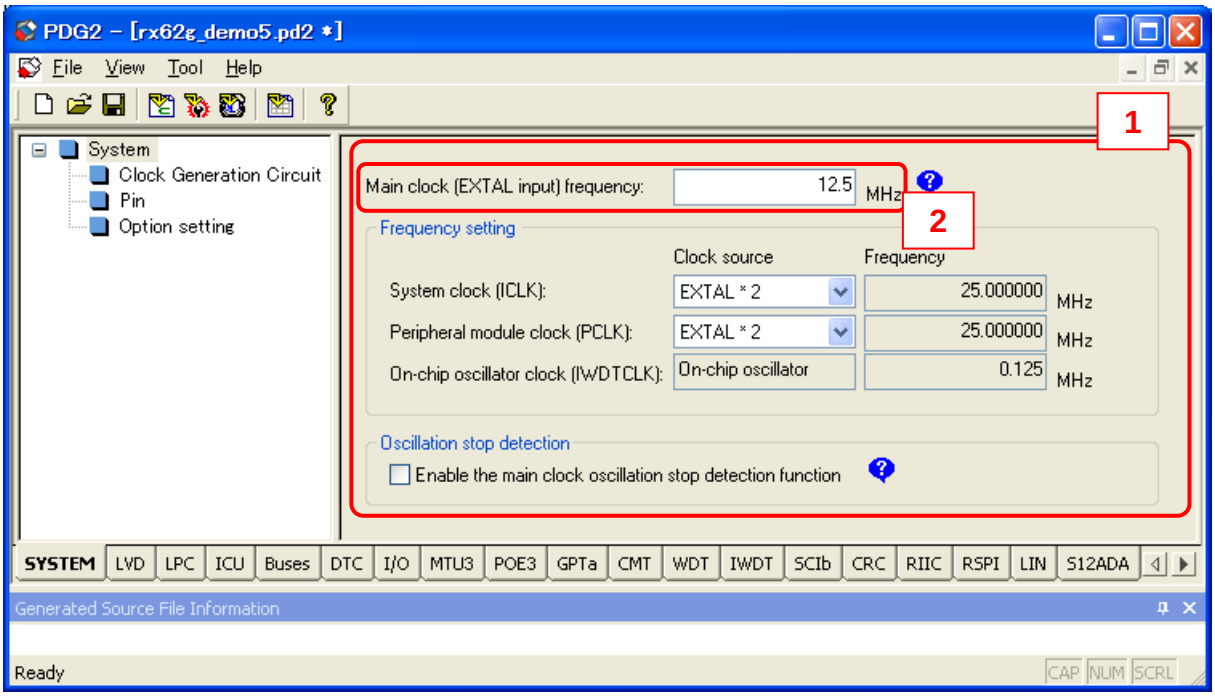
Series : RX600
Group : RX62G
Type : R5F562GAADFP

Note: If another type of chip is mounted on your RSK board, select corresponding CPU type.



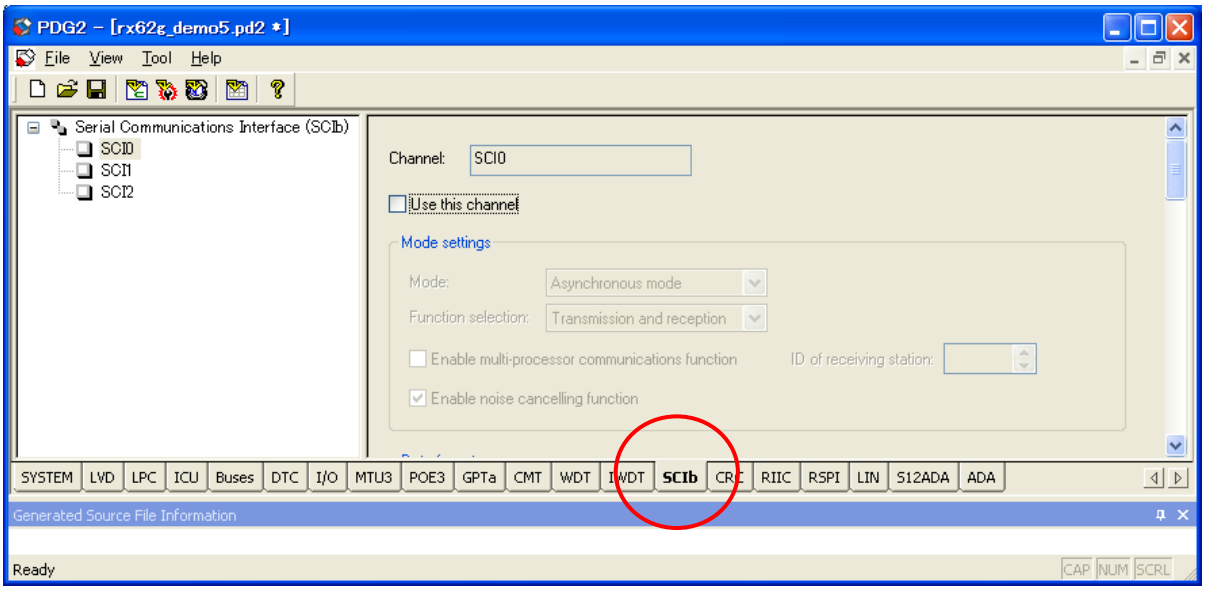
(2) Clock setting **PDG**

1. The clock setting window opens and the error icons are displayed in the initial state. For icons such as and displayed on window, refer to section 4.1(2), Initial state.
2. External clock frequency of the RSK board is 12.5 MHz. Input “12.5” into the edit box.



(3) SCIb setting **PDG**

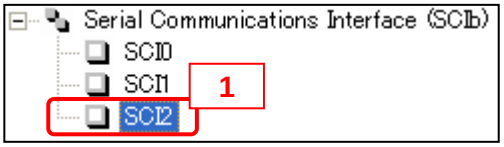
Select “SCIb” tab to open the SCIb setting window.



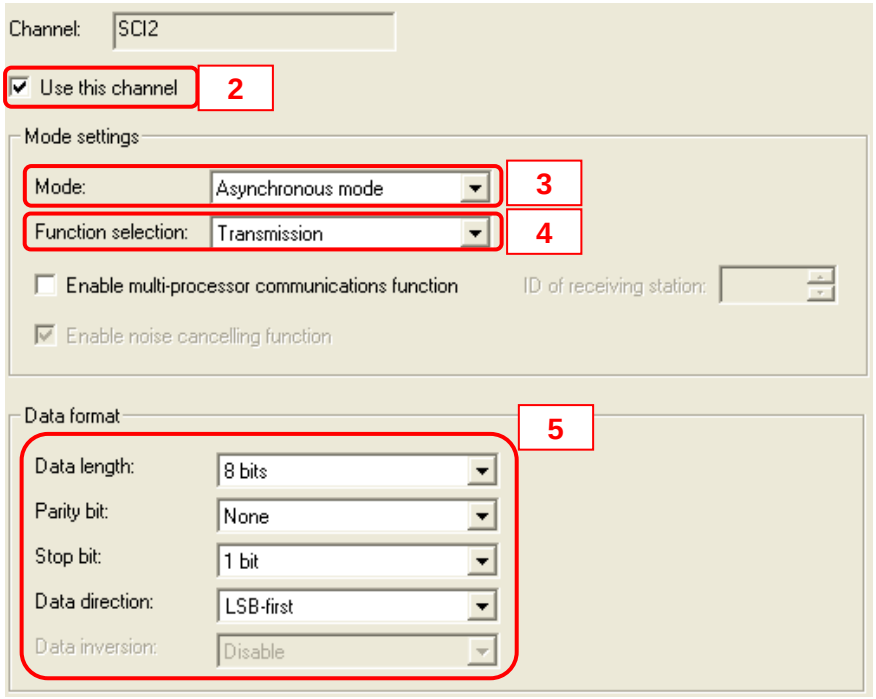
(4) SCI2 (transmitter) setting **PDG**

Make the setting for SCI2 as follows.

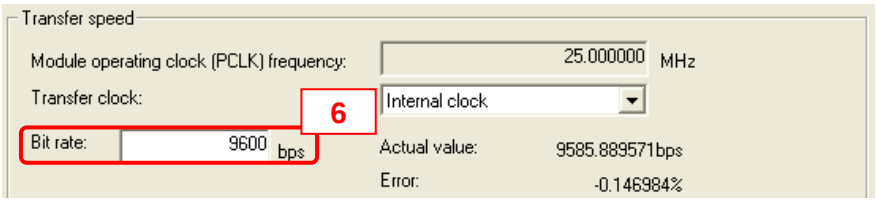
1. Select SCI2 on the tree view.



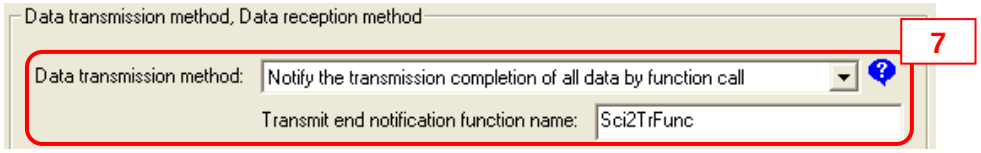
2. Check “Use this channel”.
3. Select “Asynchronous mode”.
4. Select “Transmission” for the function.
5. Leave the data format settings at the default.



6. Set the bit rate to 9,600 bps.



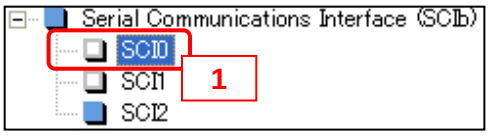
7. Select “Notify the transmission completion of all data by function call” for the data transmission method.



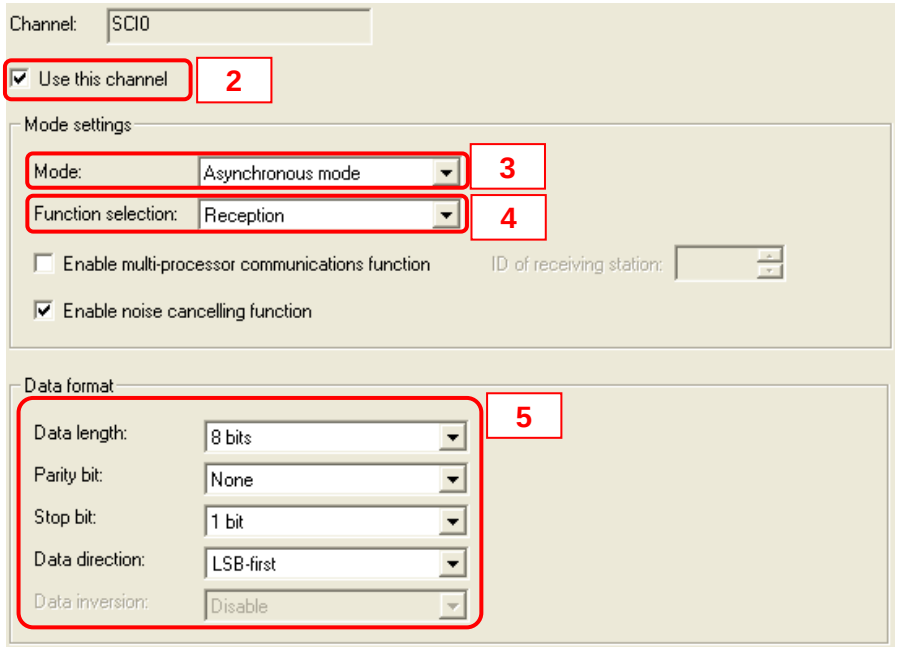
(5) SCI0 (receptor) setting **PDG**

Make the setting for SCI0 as follows.

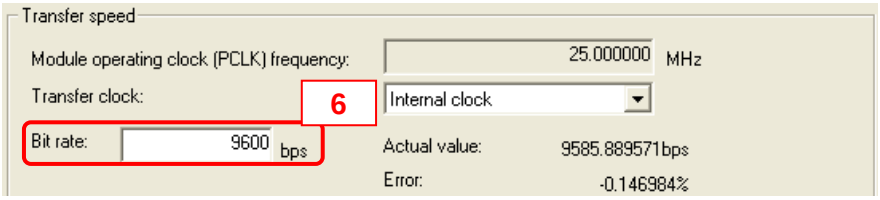
1. Select SCI0 on the tree view.



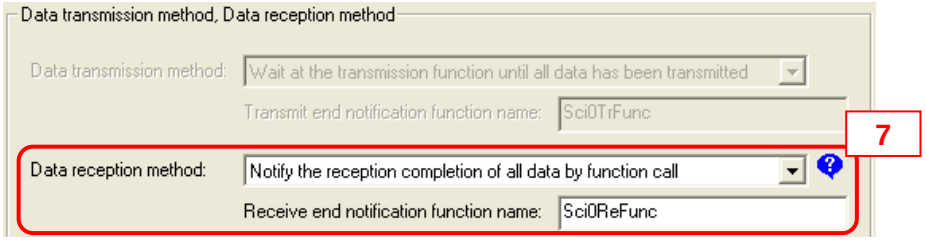
2. Check “Use this channel”.
3. Select “Asynchronous mode”.
4. Select “Reception” for the function.
5. Leave the data format settings at the default.



6. Set the bit rate to 9,600 bps.



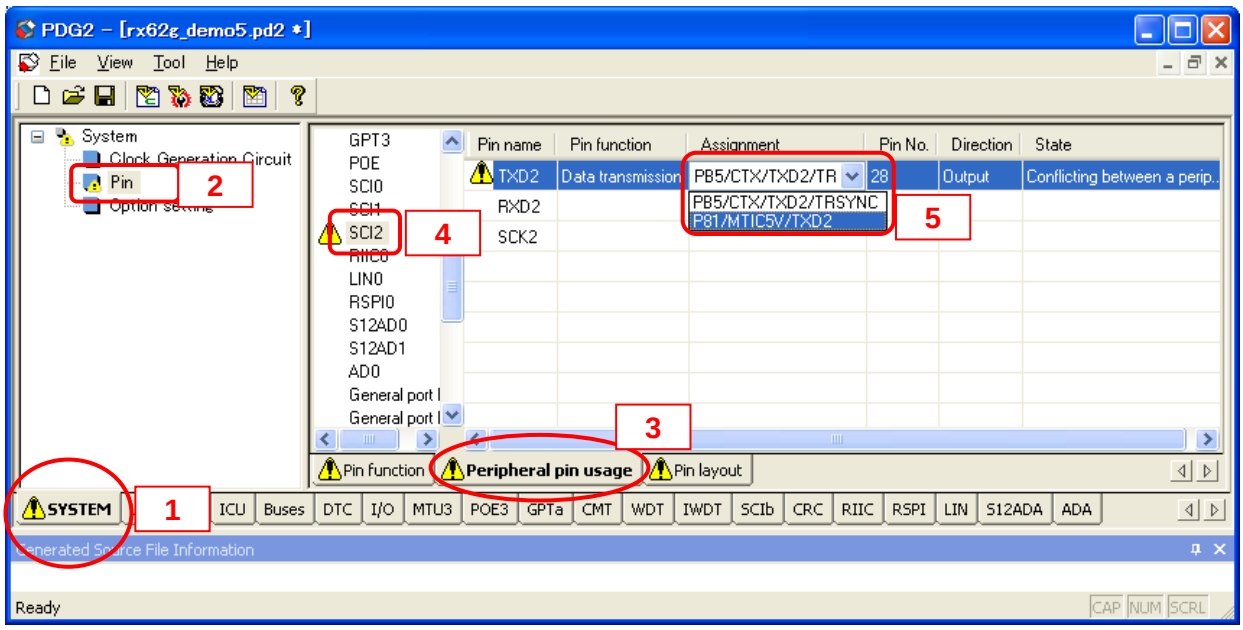
7. Select “Notify the reception completion of all data by function call” for the data reception method.



(6) Pin setting **PDG**

The TXD2 can be assigned to TXD2-A (PB5) or TXD2-B (P81). Select the pin function assignment as follows.

1. Select “SYSTEM” tab.
2. Select “Pin” on tree view.
3. Select “Peripheral pin usage” tab.
4. Select “SCI2” from the peripheral module list.
5. When the mouse pointer is placed on “Assignment” column of TXD2 line, a dropdown button is displayed. Select “P81/MTIC5V/TXD2” from the dropdown list.



(7) Generating source files **PDG**

To generate source files, click  on the tool bar. For details on generating source files, refer to section 4.1(8), Generating source files.

(8) Preparing the High-performance Embedded Workshop project **HEW**

Start the High-performance Embedded Workshop and make RX62G workspace. For details on making High-performance Embedded Workshop project, refer to section 4.1(9), Preparing the High-performance Embedded Workshop project.

PDG

(9) Adding the generated source files to the High-performance Embedded Workshop project

To add the generated source files to High-performance Embedded Workshop, click  on the tool bar. For details on adding the source files to High-performance Embedded Workshop project, refer to section 4.1(10), Adding the generated source files to the High-performance Embedded Workshop project.

(10) Making the program on High-performance Embedded Workshop **HEW**

By changing the part of “main” function, make the following program on High-performance Embedded Workshop.

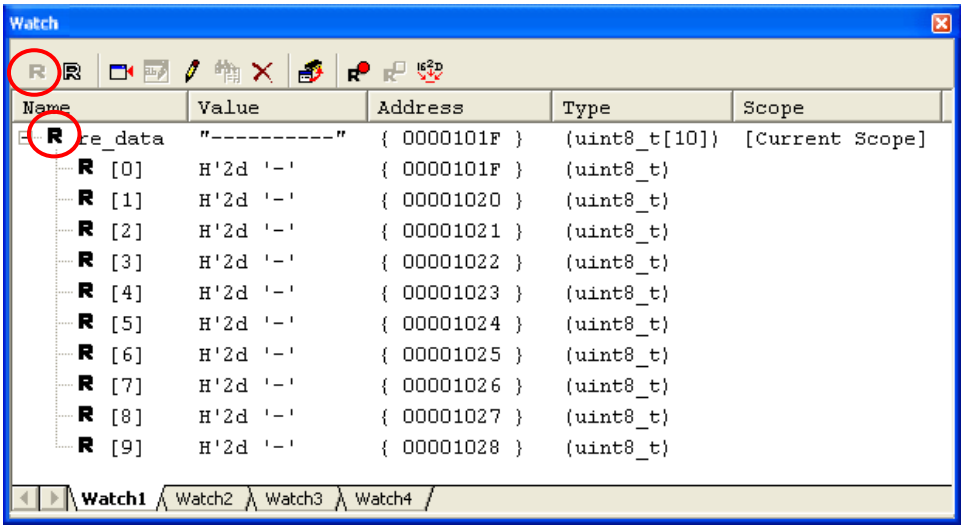
```
//Include "R_PG_<project name>.h"
#include "R_PG_rx62g_demo5.h"

//SCI2 transmission data
uint8_t tr_data[10] = "ABCDEFGHIJ";

//SCI0 reception data storage area
uint8_t re_data[10] = "----------";

void main(void)
{
      // Set up the clock
      R_PG_Clock_Set();

      // Set up the SCI2
      R_PG_SCI_Set_C2();

      // Set up the SCI0
      R_PG_SCI_Set_C0();

      // Start SCI0 reception (number of data : 10)
      R_PG_SCI_StartReceiving_C0( re_data, 10 );

      // Start SCI2 transmission (number of data : 10)
      R_PG_SCI_StartSending_C2( tr_data, 10 );

      while(1);
}

//SCI2 transmission end notification function
void Sci2TrFunc(void)
{
      //Stop SCI2 communication
      R_PG_SCI_StopCommunication_C2();
}

//SCI0 reception end notification function
void Sci0ReFunc(void)
{
      //Stop SCI0 communication
      R_PG_SCI_StopCommunication_C0();
}
```

(11) Connecting to the emulator, building the program and downloading **HEW**

Buid the program and download it. For details on connecting to the emulator, building the program, and downloading refer to section 4.1(12), connecting to the emulator, building the program and executing.

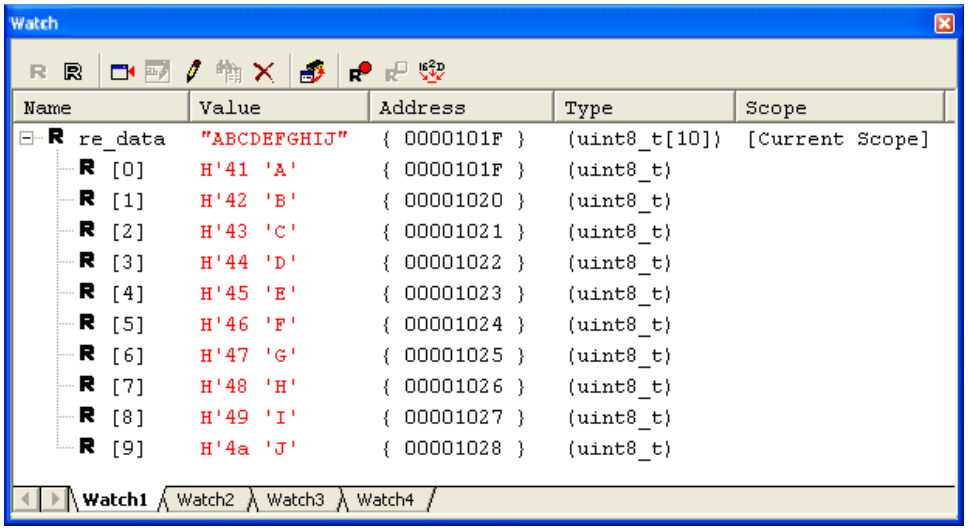
(12) Adding the variable of the reception data **HEW**

Open the Watch window and add the variable "re_data". Expand the array and set it to the real time update to monitor the variable change during execution.



(13) Executing the program and monitoring the result of the transfer **HEW**

Start the execution and check the value of “re_data” on the watch window.



5. Specification of Generated Functions

Table 5.1 shows generated functions for the RX62G.

Table 5.1 Generated Functions for the RX62G

Clock-generation circuit

Generated Function	Description
R_PG_Clock_Set	Set up the clocks
R_PG_Clock_GetMainClockStatus	Get the main clock status

Voltage Detection Circuit (LVD)

Generated Function	Description
R_PG_LVD_Set	Set up the voltage detection circuit
R_PG_LVD_GetLVDDetectionFlag	Acquire the values of the LVD detection flags

Low Power Consumption

Generated Function	Description
R_PG_LPC_Set	Set up the low power consumption functions.
R_PG_LPC_Sleep	Enter sleep mode.
R_PG_LPC_AllModuleClockStop	Enter all module clock stop mode.
R_PG_LPC_SoftwareStandby	Enter software standby mode
R_PG_LPC_DeepSoftwareStandby	Enter deep software standby mode
R_PG_LPC_IOPortRelease	Release retained I/O port state
R_PG_LPC_GetPowerOnResetFlag	Acquire the value of the power-on reset flag
R_PG_LPC_GetLVDDetectionFlag	Acquire the values of the LVD detection flags
R_PG_LPC_GetDeepSoftwareStandbyResetFlag	Acquire the value of the deep software standby reset flag
R_PD_LPC_GetDeepSoftwareStandbyCancelFlag	Acquire the value of the deep software standby cancel flag
R_PG_LPC_GetStatus	Get the status of the low power consumption functions
R_PG_LPC_WriteBackup	Write data into the backup registers
R_PG_LPC_ReadBackup	Read data from the backup registers

Interrupt controller (ICU)

Generated Function	Description
R_PG_ExtInterrupt_Set_<*interrupt type*>	Set up an external interrupt
R_PG_ExtInterrupt_Disable_<*interrupt type*>	Disable an external interrupt
R_PG_ExtInterrupt_GetRequestFlag_<*interrupt type*>	Get an external interrupt request flag
R_PG_ExtInterrupt_ClearRequestFlag_<*interrupt type*>	Clear an external interrupt request flag
R_PG_SoftwareInterrupt_Set	Set up the software interrupt
R_PG_SoftwareInterrupt_Generate	Generate the software interrupt
R_PG_FastInterrupt_Set	Set an interrupt as the fast interrupt
R_PG_Exception_Set	Set exception handlers

Buses

Generated Function	Description
R_PG_ExtBus_SetBus	Set up bus error monitoring
R_PG_ExtBus_GetErrorStatus	Acquire the status of bus error generation
R_PG_ExtBus_ClearErrorFlags	Clear the bus-error status registers

Data Transfer Controller (DTC)

Generated Function	Description
R_PG_DTC_Set	Set up the DTC
R_PG_DTC_Set_<trigger source>	Set up DTC transfer data
R_PG_DTC_Activate	Make DTC be ready for the trigger
R_PG_DTC_SuspendTransfer	Stop transfer
R_PG_DTC_GetTransmitStatus	Get transfer status
R_PG_DTC_StopModule	Shut down the DTC

I/O port

Generated Function	Description
R_PG_IO_PORT_Set_P<*port number*>	Set the I/O ports
R_PG_IO_PORT_Set_P<*port number*><*pin number*>	Set an I/O port (one pin)
R_PG_IO_PORT_Read_P<*port number*>	Read data from an I/O port register
R_PG_IO_PORT_Read_P<*port number*><*pin number*>	Read a bit from an I/O port register
R_PG_IO_PORT_Write_P<*port number*>	Write data to an I/O port data register
R_PG_IO_PORT_Write_P <*port number*><*pin number*>	Write a bit to an I/O port data register

Multi-Function Timer Pulse Unit 3 (MTU3)

Generated Function	Description
R_PG_Timer_Set_MTU_U<*unit number*>_<*channels*>	Set up the MTU
R_PG_Timer_StartCount_MTU_U<*unit number*>_C<*channel number*>(_<*phase*>)	Start the MTU count operation
R_PG_Timer_SynchronouslyStartCount_MTU_U<*unit number*>	Start the MTU count operation simultaneously
R_PG_Timer_HaltCount_MTU_U<*unit number*>_C<*channel number*>(_<*phase*>)	Halt the MTU count operation
R_PG_Timer_GetCounterValue_MTU_U<*unit number*>_C<*channel number*>	Acquire the MTU counter value
R_PG_Timer_SetCounterValue_MTU_U<*unit number*>_C<*channel number*>(_<*phase*>)	Set the counter value
R_PG_Timer_GetRequestFlag_MTU_U<*unit number*>_C<*channel number*>	Acquire the interrupt request flag
R_PG_Timer_StopModule_MTU_U<*unit number*>	Shut down the MTU unit
R_PG_Timer_GetTGR_MTU_U<*unit number*>_C<*channel number*>	Acquire the general register values
R_PG_Timer_SetTGR_<*general register*>_MTU_U<*unit number*>_C<*channel number*>	Set the general register A value
R_PG_Timer_SetBuffer_AD_MTU_U<*unit number*>_C<*channel number*>	Set A/D converter start request cycle set buffer registers
R_PG_Timer_SetBuffer_CycleData_MTU_U<*unit number*>_<*channels*>	Set the cycle buffer register
R_PG_Timer_SetOutputPhaseSwitch_MTU_U<*unit number*>_<*channels*>	Switch PWM output level
R_PG_Timer_ControlOutputPin_MTU_U<*unit number*>_<*channels*>	Enable or disable the PWM output

R_PG_Timer_SetBuffer_PWMOutputLevel_MTU_U<*unit number*>_<*channels*>	Set the PWM output level in the buffer register
R_PG_Timer_ControlBufferTransfer_MTU_U<*unit number*>_<*channels*>	Enable or disable buffer transfer from the buffer registers to the temporary registers

Port Output Enable 3 (POE3)

Generated Function	Description
R_PG_POE_Set	Set up the POE
R_PG_POE_SetHiZ_<*Timer channels*>	Place the MTU output pins in high-impedance state
R_PG_POE_GetRequestFlagHiZ_<*Timer channels*>	Acquire the high-impedance request flags
R_PG_POE_GetShortFlag_<*Timer channels*>	Acquire the MTU output short flags
R_PG_POE_ClearFlag_<*Timer channels*>	Clear the high-impedance request flags and the output short flags

General PWM Timer (GPT)

Generated Function	Description
R_PG_Timer_Set_GPT_U<*unit number*>	Set up the GPT
R_PG_Timer_Set_GPT_U<*unit number*>_C<*channel number*>	Set up the GPT channel
R_PG_Timer_StartCount_GPT_U<*unit number*>_C<*channel number*>	Start the GPT count operation
R_PG_Timer_SynchronouslyStartCount_GPT_U<*unit number*>	Start the GPT count operation of two or more channels simultaneously
R_PG_Timer_HaltCount_GPT_U<*unit number*>_C<*channel number*>	Halt the GPT count operation
R_PG_Timer_SynchronouslyHaltCount_GPT_U<*unit number*>	Halt the GPT count operation of two or more channels simultaneously
R_PG_Timer_SetGTCCR_<*GTCCR*>_GPT_U<*unit number*>_C<*channel number*>	Set the value to the compare capture register (GTCCRn n : A to F)
R_PG_Timer_GetGTCCR_GPT_U<*unit number*>_C<*channel number*>	Get the value from the compare capture registers (GTCCRA to F)
R_PG_Timer_SetCounterValue_GPT_U<*unit number*>_C<*channel number*>	Set the GPT counter value
R_PG_Timer_GetCounterValue_GPT_U<*unit number*>_C<*channel number*>	Get the GPT counter value
R_PG_Timer_SynchronouslyClearCounter_GPT_U<*unit number*>	Clear the counter of two or more channels simultaneously
R_PG_Timer_SetCycle_GPT_U<*unit number*>_C<*channel number*>	Set the value to the timer cycle setting register (GTPR)
R_PG_Timer_SetBuffer_Cycle_GPT_U<*unit number*>_C<*channel number*>	Set the value to the timer cycle setting buffer register (GTPBR)
R_PG_Timer_SetDoubleBuffer_Cycle_GPT_U<*unit number*>_C<*channel number>v*	Set the value to the timer cycle setting double-buffer register (GTPDBR)
R_PG_Timer_SetAD_GPT_U<*unit number*>_C<*channel number*>	Set the value to the A/D converter start request timing register A, B (GTADTRA, B)
R_PG_Timer_SetBuffer_AD_GPT_U<*unit number*>_C<*channel number*>	Set the value to the A/D converter start request timing buffer register A, B (GTADTBRA, GTADTBRB)
R_PG_Timer_SetDoubleBuffer_AD_GPT_U<*unit number*>_C<*channel number*>	Set the value to the A/D converter start request timing double-buffer register A, B (GTADTDBRA, GTADTDBRB)
R_PG_Timer_SetBuffer_GTDV<*U/D*>_GPT_U<*unit number*>_C<*channel number*>	Set the value to the timer dead time buffer register U, D (GTDBU, GTDBD)
R_PG_Timer_GetRequestFlag_GPT_U<*unit number*>_C<*channel number*>	Get and clear the GPT interrupt flag
R_PG_Timer_GetRequestFlag_GPT_U<*unit number*>_C<*channel number*>	Get and clear the GPT interrupt flags of LOCO count function and external trigger
R_PG_Timer_GetCounterStatus_GPT_U<*unit number*>_C<*channel number*>	Get the counter status

R_PG_Timer_BufferEnable_GPT_U<*unit number*>_C<*channel number*>	Enable the buffer operation
R_PG_Timer_BufferDisable_GPT_U<*unit number*>_C<*channel number*>	Disable the buffer operation
R_PG_Timer_Buffer_Force_GPT_U<*unit number*>_C<*channel number*>	Execute forcible buffer transfer
R_PG_Timer_CountDirection_Down_GPT_U<*unit number*>_C<*channel number*>	Set the count direction to down-counting
R_PG_Timer_CountDirection_Up_GPT_U<*unit number*>_C<*channel number*>	Set the count direction to up-counting
R_PG_Timer_SoftwareNegate_GPT_U<*unit number*>_C<*channel number*>	Control GTIOCnA and GTIOCnB pin output negation by software (n:Channel number)
R_PG_Timer_StartCount_LOCO_GPT_U<*unit number*>	Start the LOCO count
R_PG_Timer_HaltCount_LOCO_GPT_U<*unit number*>	Halt the LOCO count
R_PG_Timer_ClearCounter_LOCO_GPT_U<*unit number*>	Clear the LOCO count value register
R_PG_Timer_InitialiseCountResultValue_LOCO_GPT_U<*unit number*>	Initialize the LOCO count result registers
R_PG_Timer_GetCounterValue_LOCO_GPT_U<*unit number*>	Get the value of the LOCO count value register
R_PG_Timer_GetCounterAverageValue_LOCO_GPT_U<*unit number*>	Get the LOCO count result average value
R_PG_Timer_GetCountResultValue_LOCO_GPT_U<*unit number*>	Get the LOCO count result registers value
R_PG_Timer_SetPermissibleDeviation_LOCO_GPT_U<*unit number*>	Set the LOCO count upper/lower permissible deviation value
R_PG_Timer_StopModule_GPT_U<*unit number*>	Shut down the GPT unit

Compare Match Timer (CMT)

Generated Function	Description
R_PG_Timer_Start_CMT_U<*unit number*>_C<*channel number*>	Set up the CMT and start the count operation
R_PG_Timer_HaltCount_CMT_U<*unit number*>_C<*channel number*>	Halt the CMT count operation
R_PG_Timer_ResumeCount_CMT_U<*unit number*>_C<*channel number*>	Resume the CMT count operation
R_PG_Timer_GetCounterValue_CMT_U<*unit number*>_C<*channel number*>	Acquire the CMT counter value
R_PG_Timer_SetCounterValue_CMT_U<*unit number*>_C<*channel number*>	Set the CMT counter value
R_PG_Timer_StopModule _CMT_U<*unit number*>	Shut down the CMT unit

Watchdog Timer (WDT)

Generated Function	Description
R_PG_Timer_Start_WDT	Set up the WDT and start the count
R_PG_Timer_HaltCount_WDT	Stop the count operation
R_PG_Timer_ResetCounter_WDT	Reset the counter
R_PG_Timer_ClearOverflowFlag_WDT	Clear the counter overflow flag

Independent Watchdog Timer (IWDT)

Generated Function	Description
R_PG_Timer_Set_IWDT	Set up the IWDT
R_PG_Timer_RefreshCounter_IWDT	Refresh the counter
R_PG_Timer_GetCounterValue_IWDT	Acquire the IWDT counter value
R_PG_Timer_ClearUnderflowFlag_IWDT	Acquire and clear the underflow flag

Serial Communications Interface (SCI)

Generated Function	Description
R_PG_SCI_Set_C<*channel number*>	Set up a SCI channel
R_PG_SCI_StartSending_C<*channel number*>	Start the data transmission
R_PG_SCI_SendAllData_C<*channel number*>	Transmit all data
R_PG_SCI_GetSentDataCount_C<*channel number*>	Acquire the number of transmitted data

R_PG_SCI_StartReceiving_C<*channel number*>	Start the data reception
R_PG_SCI_ReceiveAllData_C<*channel number*>	Receive all data
R_PG_SCI_StopCommunication_C<*channel number*>	Stop transmission and reception
R_PG_SCI_GetReceivedDataCount_C<*channel number*>	Acquire the number of received data
R_PG_SCI_GetReceptionErrorFlag_C<*channel number*>	Get the serial reception error flag
R_PG_SCI_GetTransmitStatus_C<*channel number*>	Get the state of transmission
R_PG_SCI_SendTargetStationID_C<*channel number*>	Transmits the ID code of the receiving station
R_PG_SCI_ReceiveStationID_C<*channel number*>	Receives the ID code matches the ID of the receiving station itself
R_PG_SCI_StopModule_C<*channel number*>	Shut down a SCI channel
R_PG_SCI_ControlClockOutput_C<*channel number*>	Control the SCKn pin output

CRC Calculator (CRC)

Generated Function	Description
R_PG_CRC_Set	Set up CRC calculator
R_PG_CRC_InputData	Input a data to CRC calculator
R_PG_CRC_GetResult	Get the the result of calculation
R_PG_CRC_StopModule	Shut down CRC Calculator

I2C Bus Interface (RIIC)

Generated Function	Description
R_PG_I2C_Set_C<*channel number*>	Set up the I2C bus interface channel
R_PG_I2C_MasterReceive_C<*channel number*>	Master data reception
R_PG_I2C_MasterReceiveLast_C<*channel number*>	Complete a master reception process
R_PG_I2C_MasterSend_C<*channel number*>	Master data transmission
R_PG_I2C_MasterSendWithoutStop_C<*channel number*>	Master data transmission (No stop condition)
R_PG_I2C_GenerateStopCondition_C<*channel number*>	Generate a stop condition
R_PG_I2C_GetBusState_C<*channel number*>	Get the bus status
R_PG_I2C_SlaveMonitor_C<*channel number*>	Slave bus monitor
R_PG_I2C_SlaveSend_C<*channel number*>	Slave data transmission
R_PG_I2C_GetDetectedAddress_C<*channel number*>	Get the detected address
R_PG_I2C_GetTR_C<*channel number*>	Get the transmit/receive mode
R_PG_I2C_GetEvent_C<*channel number*>	Get the detected event
R_PG_I2C_GetReceivedDataCount_C<*channel number*>	Acquires the count of transmitted data
R_PG_I2C_GetSentDataCount_C<*channel number*>	Acquires the count of received data
R_PG_I2C_Reset_C<*channel number*>	Reset the bus
R_PG_I2C_StopModule_C<*channel number*>	Shut down the I2C bus interface channel

Serial Peripheral Interface (RSPI)

Generated Function	Description
R_PG_RSPI_Set_C<*channel number*>	Set up a RSPI channel
R_PG_RSPI_SetCommand_C<*channel number*>	Set commands
R_PG_RSPI_StartTransfer_C<*channel number*>	Start the data transfer
R_PG_RSPI_TransferAllData_C<*channel number*>	Transfer all data
R_PG_RSPI_GetStatus_C<*channel number*>	Acquire the transfer status

R_PG_RSPI_GetError_C<*channel number*>	Acquire the error flags
R_PG_RSPI_GetCommandStatus_C<*channel number*>	Acquire the command status
R_PG_RSPI_StopModule_C<*channel number*>	Shut down a RSPI channel
R_PG_RSPI_LoopBack<*loopback mode*>_C<*channel number*>	Set loopback mode

12-Bit A/D Converter (S12ADA)

Generated Function	Description
R_PG_ADC_12_Set_S12AD<*unit number*>	Set up the 12-Bit A/D Converter
R_PG_ADC_12_Set	Set up the programmable gain amplifier
R_PG_ADC_12_StartConversionSW_S12AD<*unit number*>	Start A/D conversion (Software trigger)
R_PG_ADC_12_StopConversion_S12AD<*unit number*>	Stop A/D conversion
R_PG_ADC_12_GetResult_S12AD<*unit number*>	Acquire the result of A/D conversion
R_PG_ADC_12_GetResult_SelfDiag_S12AD<*unit number*>	Acquire the result of A/D conversion (Self-diagnosis)
R_PG_ADC_12_StopModule_S12AD<*unit number*>	Shut down the 12-Bit A/D converter

10-Bit A/D Converter (ADA)

Generated Function	Description
R_PG_ADC_10_Set_AD<*unit number*>	Set up the 10-Bit A/D Converter (ADa)
R_PG_ADC_10_StartConversionSW_AD<*unit number*>	Start A/D conversion (software trigger)
R_PG_ADC_10_StopConversion_AD<*unit number*>	Stop A/D conversion
R_PG_ADC_10_GetResult_AD<*unit number*>	Get the result of A/D conversion
R_PG_ADC_10_SetSelfDiag_VREF_<*voltage*>_AD<*unit number*>	Set up the A/D self-diagnostic function
R_PG_ADC_10_StopModule_AD<*unit number*>	Shut down the 10-Bit A/D Converter (ADa)

5.1 Clock-Generation Circuit

5.1.1 R_PG_Clock_Set

Definition: bool R_PG_Clock_Set(void)

Description: Set up the clocks

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Clock.c

RPDL function: R_CGC_Set

Details:

- Sets registers in the clock-generation circuit and multiplication ratios to derive the system clock (ICLK) and peripheral module clock (PCLK).
- Sets the oscillation stop detection function.

Example:

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

void func(void)
{
    //Set the clock-generation circuit.
    R_PG_Clock_Set();
}
```

5.1.2 R_PG_Clock_GetMainClockStatus

Definition: bool R_PG_Clock_GetMainClockStatus(bool * stop)

Description: Get the main clock oscillation stop detection flag

Conditions for output: The main clock oscillator stop detection function is enabled

Parameter

bool * stop	The address of the storage area for the main clock oscillation stop flag

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output: R_PG_Clock.c

RPDL function: R_CGC_GetStatus

Details

- This function gets the main clock oscillation stop detection flag.
- To generate the NMI when the main clock oscillator stop is detected, enable the oscillation stop detection interrupt through the NMI settings in GUI. The NMI can be set up by the function R_PG_ExtInterrupt_Set_NMI.

Example

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

bool main_stop;

void func(void)
{
    // Get the main clock oscillation stop detection flag
    R_PG_Clock_GetMainClockStatus(&main_stop);
}
```

5.2 Voltage Detection Circuit (LVD)

5.2.1 R_PG_LVD_Set

Definition: bool R_PG_LVD_Set (void)

Description: Set up the voltage detection circuit.

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_LVD.c

RPDL function: R_LVD_Control

Details:

- This function sets the operation (internal reset or interrupt) when low voltage is detected.
- Both LVD1 and LVD2 can be set up in one function call.
- When an interrupt is selected as the operation in case of low voltage detection, NMI must be set up. To generate the NMI when low voltage is detected, enable the power-voltage falling detection interrupt through the NMI settings in GUI. The NMI can be set up by the function R_PG_ExtInterrupt_Set_NMI.
- Use R_PG_LVD_GetLVDDetectionFlag to acquire the low voltage detection flags (LVD1 and LVD2).

Example:

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

void func(void)
{
    // Set up the voltage detection circuit.
    R_PG_LVD_Set (void);
}
```

5.2.2 R_PG_LVD_GetLVDDetectionFlag

Definition: bool R_PG_LVD_GetLVDDetectionFlag (bool * lvd1, bool * lvd2)

Description: Acquire the values of the LVD detection flags.

Parameter

bool * lvd1	The address of the storage area for the LVD1 detection flag
bool * lvd2	The address of the storage area for the LVD2 detection flag

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output: R_PG_LVD.c

RPDL function: R_LPC_GetStatus

Details

- This function acquires the value of the LVD detection flag.
- Specify 0 for a flag that is not required.

Example

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

bool lvd1;
bool lvd2;

void func(void)
{
    // Acquire the LVD1 and LVD2 flags.
    R_PG_LVD_GetLVDDetectionFlag ( &lvd1, &lvd2);

    if( lvd1 ){
        //Processing when the LVD1 is detected
    }
    if( lvd2 ){
        //Processing when the LVD2 is detected
    }
}
```

5.3 Low Power Consumption

5.3.1 R_PG_LPC_Set

Definition: bool R_PG_LPC_Set (void)

Description: Set up the low power consumption functions.

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_LPC.c

RPDL function: R_LPC_Create

Details:

- This function configures the low power conditions.

Example:

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

void func(void)
{
    // Set up the low power consumption functions.
    R_PG_LPC_Set (void);
}
```

5.3.2 R_PG_LPC_Sleep

Definition bool R_PG_LPC_Sleep (void)

Description Enter sleep mode.

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_LPC.c

RPDL function R_LPC_Control

Details

- This function set the system to sleep mode.

Example

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

void func(void)
{
    // Enter sleep mode.
    R_PG_LPC_Sleep(void);
}
```

5.3.3 R_PG_LPC_AllModuleClockStop

Definition bool R_PG_LPC_AllModuleClockStop (void)

Description Enter all module clock stop mode.

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_LPC.c

RPDL function R_LPC_Control

Details

- This function sets the system to all module clock stop mode.

Example

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

void func(void)
{
    // Enter all module clock stop mode.
    R_PG_LPC_AllModuleClockStop (void);
}
```

5.3.4 R_PG_LPC_SoftwareStandby

Definition bool R_PG_LPC_SoftwareStandby(void)

Description Enter software standby mode.

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_LPC.c

RPDL function R_LPC_Control

Details
- This function set the system to software standby mode.
- Call R_PG_LPC_Set before calling this function to set the operation during software standby mode.

Example

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

void func(void)
{
    // Set up the low power consumption functions.
    R_PG_LPC_Set (void);

    // Enter software standby mode.
    R_PG_LPC_SoftwareStandby (void);
}
```

5.3.5 R_PG_LPC_DeepSoftwareStandby

Definition bool R_PG_LPC_DeepSoftwareStandby(void)

Description Enter deep software standby mode.

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_LPC.c

RPDL function R_LPC_Control

Details

- This function set the system to deep software standby mode.
- Call R_PG_LPC_Set before calling this function to set the operation during deep software standby mode and release triggers.
- The deep software standby cancel flag is set to 1 when a cancel request is generated in any mode. In this function, the deep software standby cancel flag is not cleared before entering deep software standby mode. Clear the deep software standby cancel flag before calling this function by R_PD_LPC_GetDeepSoftwareStandbyCancelFlag.

Example

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

void func(void)
{
    // Set up the low power consumption functions.
    R_PG_LPC_Set (void);

    // Clear deep software standby cancel flag.
    R_PD_LPC_GetDeepSoftwareStandbyCancelFlag(0,0,0,0);

    // Enter deep software standby mode.
    R_PG_LPC_DeepSoftwareStandby (void);
}
```

5.3.6 R_PG_LPC_IOPortRelease

Definition	bool R_PG_LPC_IOPortRelease (void)
Description	Release retained I/O port state.
Conditions for output	On the GUI, [Release retained port state when 0 is written to the IOKEEP bit after release from deep software standby mode] is selected for the setting of [I/O port state retention].
Parameter	None

Return value

true	Setting was made correctly
false	Setting failed

File for output	R_PG_LPC.c
RPDL function	R_LPC_Control

Details

- This function releases I/O ports from the retention state after the system is releasd from deep software standby mode.

Example

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"
void func(void)
{
    // Release I/O ports from the retention state
    R_PG_LPC_IOPortRelease(void);
}
```

5.3.7 R_PG_LPC_GetPowerOnResetFlag

Definition: bool R_PG_LPC_GetPowerOnResetFlag (bool *reset)

Description: Acquire the value of the power-on reset flag.

Parameter

bool *reset	The address of the storage area for the power-on reset flag

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output: R_PG_LPC.c

RPDL function: R_LPC_GetStatus

Details

- This function acquires the value of the power-on reset flag.
- The RSTSR.LVD1F (LVD1 detection flag), RSTSR.LVD2F(LVD2 detection flag), RSTSR.DPSRSTF (deep software standby reset flag) and DPSIFR(deep software standby cancel request flags) are cleared by calling this function. Use R_PG_LPC_GetStatus instead of this function to get these flags simultaneously if
- needed.
  RSTSR.PORF(power-on reset flag) is only initialized by a pin reset.

Example

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

bool reset;

void func(void)
{
    // Acquire the power-on reset flags.
    R_PG_LPC_GetPowerOnResetFlag( &reset );

    if( reset ){
        // Processing when the power-on reset is detected
    }
}
```

5.3.8 R_PG_LPC_GetLVDDetectionFlag

Definition: bool R_PG_LPC_GetLVDDetectionFlag (bool * lvd1, bool * lvd2)

Description: Acquire the values of the LVD detection flags.

Parameter

bool * lvd1	The address of the storage area for the LVD1 detection flag
bool * lvd2	The address of the storage area for the LVD2 detection flag

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output: R_PG_LPC.c

RPDL function: R_LPC_GetStatus

Details

- This function acquires the value of the LVD detection flags.
- Specify the address of storage area for the flags to be acquired.
- Specify 0 for a flag that is not required.
- The RSTSR.LVD1F (LVD1 detection flag), RSTSR.LVD2F(LVD2 detection flag), RSTSR.DPSRSTF (deep software standby reset flag) and DPSIFR(deep software standby cancel request flags) are cleared by calling this function. Use R_PG_LPC_GetStatus instead of this function to get these flags simultaneously if needed.

Example

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

bool lvd1;
bool lvd2;

void func(void)
{
    // Acquire the LVD1 and LVD2 flags.
    R_PG_LPC_GetLVDDetectionFlag ( &lvd1, &lvd2);

    if( lvd1 ){
        //Processing when the LVD1 is detected
    }
    if( lvd2 ){
        //Processing when the LVD2 is detected
    }
}
```

5.3.9 R_PG_LPC_GetDeepSoftwareStandbyResetFlag

Definition: bool R_PG_LPC_GetDeepSoftwareStandbyResetFlag(bool *reset)

Description: Acquire the value of the deep software standby reset flag.

Parameter

bool *reset	The address of the storage area for the deep software standby reset flag

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output: R_PG_LPC.c

RPDL function: R_LPC_GetStatus

Details

- This function acquires the value of the deep software standby reset flag.
- The RSTSR.LVD1F (LVD1 detection flag), RSTSR.LVD2F(LVD2 detection flag), RSTSR.DPSRSTF (deep software standby reset flag) and DPSIFR(deep software standby cancel request flags) are cleared by calling this function. Use R_PG_LPC_GetStatus instead of this function to get these flags simultaneously if needed.

Example

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

bool reset;

void func(void)
{
    // Acquire the deep software standby reset flag.
    R_PG_LPC_GetDeepSoftwareStandbyResetFlag ( &reset);

    if( reset ){
        //Processing when the deep software standby reset is detected
    }
}
```

5.3.10 R_PD_LPC_GetDeepSoftwareStandbyCancelFlag

Definition

bool R_PD_LPC_GetDeepSoftwareStandbyCancelFlag
(bool *irq0, bool *irq1, bool *lvd, bool *nmi)

Description

Acquire the value of the deep software standby cancel request flags.

Parameter

bool *irq0	The address of the storage area for the flag of cancel request by IRQ0
bool *irq1	The address of the storage area for the flag of cancel request by IRQ1
bool *lvd	The address of the storage area for the flag of cancel request by LVD
bool *nmi	The address of the storage area for the flag of cancel request by NMI

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output

R_PG_LPC.c

RPDL function

R_LPC_GetStatus

Details

- This function acquires the value of the deep software standby cancel request flags.
- Specify the address of storage area for the flags to be acquired.
- Specify 0 for a flag that is not required.
- The RSTSR.LVD1F (LVD1 detection flag), RSTSR.LVD2F(LVD2 detection flag), RSTSR.DPSRSTF (deep software standby reset flag) and DPSIFR(deep software standby cancel request flags) are cleared by calling this function. Use R_PG_LPC_GetStatus instead of this function to get these flags simultaneously if needed.

Example

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"
bool irq0;
bool nmi;
void func(void)
{
      // Acquire the deep software standby cancel request flags (IQR0-A and NMI)
      R_PD_LPC_GetDeepSoftwareStandbyCancelFlag ( &irq0, 0, 0, &nmi );

      if( irq0 ){
        //Processing when the deep software standby cancel request form IRQ0-A is detected
      }
      if( nmi ){
        // Processing when the deep software standby cancel request form NMI is detected
      }
}
```

5.3.11 R_PG_LPC_GetStatus

Definition: bool R_PG_LPC_GetStatus(uint16_t *data)

Description: Get the status of the low power consumption functions.

Parameter

uint16_t *data	The address of the storage area for the status data

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output: R_PG_LPC.h

RPDL function: R_LPC_GetStatus

Details

- This function acquires the reset status and deep software standby cancel request flags.
- When calling this function, the function of RPDL R_PG_LPC_GetStatus is called directly.
- The status flags shall be stored in the format below.

b15	b14-b11	b10	b9	b8
Reset status (RSTSR) (0: not detected; 1: detected)				
Deep software reset	0	LVD2	LVD1	Power-on reset

b7	b6	b5	b4	b3	b2	b1	b0
Deep software standby cancel request detection (DPSIFR) (0: not detected; 1: detected)							
NMI	0	0	LVD	0	0	IRQ1-A	IRQ0-A

- The RSTSR(LVD detection flags, deep software standby reset flag) and DPSIFR(deep software standby cancel request flags) are cleared by calling this function.
- RSTSR.PORF(power-on reset flag) is only initialized by a pin reset.

Example

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"
uint16_t data;
void func(void)
{
    // Acquire the LPC status
    R_PG_LPC_GetStatus( &data );

    //Has deep software standby reset been detected?
    if( (data >> 15) & 0x1 ){
        if( (data >> 7) &0x1){
            // Processing when the deep software standby is canceled by NMI
        }
        else if( data &0x1){
            // Processing when the deep software standby is canceled by IRQ0-A
        }
    }
}
```

5.3.12 R_PG_LPC_WriteBackup

Definition: bool R_PG_LPC_WriteBackup (uint8_t * data, uint8_t count)

Description: Write data into the deep standby backup registers.

Parameter

uint8_t * data	The start address of data to be written to the backup area.
uint8_t count	The number of bytes to be written to the backup area. Valid from 1 to 32.

Return value

true	Setting was made correctly
false	Setting failed

File for output: R_PG_LPC.h

RPDL function: R_LPC_WriteBackup

Details

- Writes data into the deep standby backup registers.
- When calling this function, the function of RPDL R_LPC_WriteBackup is called directly.

Example

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

uint8_t w_data[]=”ABCDEFG”;
uint8_t r_data[]=”-------”;

void func1(void)
{
    // Set up the low power consumption functions.
    R_PG_LPC_Set (void);

    // Write data into the deep standby backup registers
    R_PG_LPC_WriteBackup( w_data, 7 );

    // Enter deep software standby mode.
    R_PG_LPC_DeepSoftwareStandby (void);
}

void func2(void)
{
    // Read data from the deep standby backup registers
    R_PG_LPC_ReadBackup( r_data, 7 );
}
```

5.3.13 R_PG_LPC_ReadBackup

Definition: bool R_PG_LPC_ReadBackup (uint8_t * data, uint8_t count)

Description: Read data from the deep standby backup registers.

Parameter

uint8_t * data	The start address of the storage area for the data read from the backup area.
uint8_t count	The number of bytes to be read from the backup area. Valid from 1 to 32.

Return value

true	Acquisition succeeded.
false	Acquisition failed.

File for output: R_PG_LPC.h

RPDL function: R_LPC_ReadBackup

Details

- Reads data from the deep standby backup registers.
- When calling this function, the function of RPDL R_LPC_ReadBackup is called directly.

Example

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

uint8_t w_data[]=”ABCDEFG”;
uint8_t r_data[]=”-------”;

void func1(void)
{
    // Set up the low power consumption functions.
    R_PG_LPC_Set (void);

    // Write data into the deep standby backup registers
    R_PG_LPC_WriteBackup( w_data, 7 );

    // Enter deep software standby mode.
    R_PG_LPC_DeepSoftwareStandby (void);
}

void func2(void)
{
    // Read data from the deep standby backup registers
    R_PG_LPC_ReadBackup( r_data, 7 );
}
```

5.4 Interrupt Controller (ICU)

5.4.1 R_PG_ExtInterrupt_Set_<*interrupt type*>

Definition bool R_PG_ExtInterrupt_Set_<*interrupt type*> (void)
<*interrupt type*>: IRQ0 to IRQ7 or NMI

Description Set up an external interrupt

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_ExtInterrupt_<*interrupt type*>.c
<*interrupt type*>: IRQ0 to IRQ7 or NMI

RPDL function R_INTC_CreateExtInterrupt

Details

- Enables an external interrupt (IRQ0 to IRQ7 or the NMI) and sets the input direction and input buffer for the pins to be used for the external interrupt signal. For IRQn, the pin to be used (IRQn-A/B/C) is set according to the selection in the [Peripheral Pin Usage] window.
- When the name of the interrupt notification function has been specified in the GUI, if an interrupt occurs in the CPU, the function having the specified name will be called. Create the interrupt notification function as follows:
  void <*name of the interrupt notification function*> (void)
  For the interrupt notification function, note the contents of 5.21, Notes on Notification Functions.
- If the interrupt propriety level is set to 0 in the GUI, an interrupt handler will not be called even when the external interrupt is input. The request flag can be acquired by calling R_PG_ExtInterrupt_GetRequestFlag_<*interrupt type*> and the flag can be cleared by R_PG_ExtInterrupt_ClearRequestFlag_<*interrupt type*>.

Example1 A case where Irq0IntFunc has been specified as the name of an interrupt notification function:

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

void func(void)
{
      //Set IRQ0.
      R_PG_ExtInterrupt_Set_IRQ0();
}

//IRQ0 notification function
void Irq0IntFunc (void)
{
      func_irq0();      //Processing of IRQ0
}
```

Example2 A case where the interrupt propriety level is set to 0:

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

void func(void)
{
    bool flag;

    //Set IRQ0.
    R_PG_ExtInterrupt_Set_IRQ0();

    do{
        //Acquire the interrupt request flag for IRQ0.
        R_PG_ExtInterrupt_GetRequestFlag_IRQ0( &flag );
    }while( ! flag )

    func_irq0();        //Processing of IRQ0

    //Clear the interrupt request flag for IRQ0.
    R_PG_ExtInterrupt_ClearRequestFlag_IRQ0();
}
```

5.4.2 R_PG_ExtInterrupt_Disable_<*interrupt type*>

Definition bool R_PG_ExtInterrupt_Disable_<*interrupt type*> (void)
<*interrupt type*>: IRQ0 to IRQ7

Description Disable an external interrupt

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_ExtInterrupt_<*interrupt type*>.c
<*interrupt type*>: IRQ0 to IRQ7

RPDL function R_INTC_ControlExtInterrupt

Details
- Disables an external interrupt (IRQ0 to IRQ7).
- Settings of the input/output direction and input buffer for the pin being used for the external interrupt signal are retained.

Example A case where Irq0IntFunc has been specified as the name of an interrupt notification function:

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

void func(void)
{
      //Set IRQ0.
      R_PG_ExtInterrupt_Set_IRQ0();
}

//External interrupt (IRQ0) notification function
void Irq0IntFunc (void)
{
      //Disable IRQ0.
      R_PG_ExtInterrupt_Disable_IRQ0();

      func_irq0();      //Processing of IRQ0
}
```

5.4.3 R_PG_ExtInterrupt_GetRequestFlag_<*interrupt type*>

Definition bool R_PG_ExtInterrupt_GetRequestFlag_<*interrupt type*> (bool * flag)
<*interrupt type*>: IRQ0 to IRQ7 or NMI

Description Get an external interrupt request flag

Parameter

bool * flag	The address of the storage area for the interrupt request flag

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output R_PG_ExtInterrupt_<*interrupt type*>.c
<*interrupt type*>: IRQ0 to IRQ7 or NMI

RPDL function R_INTC_GetExtInterruptStatus

Details
- Acquires the interrupt request flag for an external interrupt (IRQ0 to IRQ7 or the NMI). When an interrupt is requested, ‘true’ is entered in the specified destination for storage of the flag’s value.

Example A case where the interrupt propriety level is set to 0:

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

void func(void)
{
    bool flag;

    //Set IRQ0.
    R_PG_ExtInterrupt_Set_IRQ0();

    do{
        //Acquire the interrupt request flag for IRQ0.
        R_PG_ExtInterrupt_GetRequestFlag_IRQ0( &flag );
    }while( ! flag )

    func_irq0();        //Processing of IRQ0

    //Clear the interrupt request flag for IRQ0.
    R_PG_ExtInterrupt_ClearRequestFlag_IRQ0();
}
```

5.4.4 R_PG_ExtInterrupt_ClearRequestFlag_<*interrupt type*>

Definition bool R_PG_ExtInterrupt_ClearRequestFlag_<*interrupt type*> (void)
<*interrupt type*>: IRQ0 to IRQ7 or NMI

Description Clear an external interrupt request flag

Parameter None

Return value

true	Clearing succeeded
false	Clearing failed

File for output R_PG_ExtInterrupt_<*interrupt type*>.c
<*interrupt type*>: IRQ0 to IRQ7 or NMI

RPDL function R_INTC_ControlExtInterrupt

Details
- Clears the interrupt request flag for an external interrupt (IRQ0 to IRQ7 or NMI).
- If the level-sensitive interrupt is selected, the interrupt request flag is cleared when high-level is input to the interrupt pin. The request flag of level-sensitive interrupt cannot be cleared by this function.

Example A case where the interrupt propriety level is set to 0:

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

void func(void)
{
    bool flag;

    //Set IRQ0.
    R_PG_ExtInterrupt_Set_IRQ0();

    do{
        //Acquire the interrupt request flag for IRQ0.
        R_PG_ExtInterrupt_GetRequestFlag_IRQ0( &flag );
    }while( ! flag )

    func_irq0();        //Processing of IRQ0

    //Clear the interrupt request flag for IRQ0.
    R_PG_ExtInterrupt_ClearRequestFlag_IRQ0();
}
```

5.4.5 R_PG_SoftwareInterrupt_Set

Definition: bool R_PG_SoftwareInterrupt_Set(void)

Description: Set up the software interrupt

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_SoftwareInterrupt.c

RPDL function: R_INTC_CreateSoftwareInterrupt

Details:

- Sets up the software interrupt.
- The software interrupt cannot be generated by calling this function. To generate the software interrupt, call R_PG_SoftwareInterrupt_Generate.

Example: A case where SwIntFunc was specified as the name of the software interrupt notification function in the GUI.

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"
void SwIntFunc(void);

void func(void)
{
    //Set up the software interrupt
    R_PG_SoftwareInterrupt_Set();

    //Generate the software interrupt
    R_PG_SoftwareInterrupt_Generate();
}

void SwIntFunc(void)
{
    //Processing of software interrupt
}
```

5.4.6 R_PG_SoftwareInterrupt_Generate

Definition: bool R_PG_SoftwareInterrupt_Generate(void)

Description: Generate the software interrupt

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_SoftwareInterrupt.c

RPDL function: R_INTC_Write

Details:

- Generates the software interrupt.
- Call R_PG_SoftwareInterrupt_Set before calling this function to set up the software interrupt.

Example: SwIntFunc was specified as the name of the software interrupt function in the GUI.

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"
void SwIntFunc(void);

void func(void)
{
    //Set up the software interrupt
    R_PG_SoftwareInterrupt_Set();

    //Generate the software interrupt
    R_PG_SoftwareInterrupt_Generate();
}

void SwIntFunc(void)
{
    //Processing of software interrupt
}
```

5.4.7 R_PG_FastInterrupt_Set

Definition: bool R_PG_FastInterrupt_Set (void)

Description: Set up the fast interrupt

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_FastInterrupt.c

RPDL function: R_INTC_CreateFastInterrupt

Details:

- Sets the interrupt source specified in the GUI as the fast interrupt. The specified interrupt source is not set or enabled. The interrupt source to be set as the fast interrupt must be set and enabled by the functions for the peripheral module.
- This function uses an unconditional trap instruction (BRK) to set the fast-interrupt vector register (FINTV). If interrupts are disabled (the interrupt enable bit (I) of the processor status word is 0), this function will be locked.
- The interrupt handler that is specified as a fast interrupt will be compiled as a fast interrupt handler by specifying fint in #pragma interrupt declaration.

Example: A case where IRQ0 has been specified as the fast interrupt in the GUI:

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

void func(void)
{
    //Set IRQ0 as the fast interrupt.
    R_PG_FastInterrupt_Set ();

    //Set IRQ0.
    R_PG_ExtInterrupt_Set_IRQ0();
}
```

5.4.8 R_PG_Exception_Set

Definition: bool R_PG_Exception_Set (void)

Description: Set the exception handlers

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Exception.c

RPDL function: R_INTC_CreateExceptionHandlers

Details:

- Sets the exception notification functions. If an exception for which the name of the exception notification function was specified in the GUI occurs after this function is called, the function with the specified name will be called.
  Create the exception notification function as follows:
  void <*name of the exception notification function*> (void)
  For the exception notification function, note the contents of 5.21, Notes on Notification Functions.

Example: A case where the following exception notification functions have been set in the GUI:
Privileged instruction exception: PrivInstExcFunc
Undefined instruction exception: UndefInstExcFunc
Floating-point exception: FpExcFunc

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

void func(void)
{
      //Set the exception handlers.
      R_PG_Exception_Set();
}

void PrivInstExcFunc(){
      func_pi_excep();       //Processing in response to a privileged instruction exception
}

void UndefInstExcFunc (){
      func_ui_excep();       //Processing in response to an undefined instruction exception
}

void FpExcFunc (){
      funct_fp_excep();       //Processing in response to a floating-point exception
}
```

5.5 Buses

5.5.1 R_PG_ExtBus_SetBus

Definition: bool R_PG_ExtBus_SetBus(void)

Description: Set up the bus error monitoring

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_ExtBus.c

RPDL function: R_BSC_Create

Details:

- Sets up the bus error monitoring.
- The bus error interrupt is set by this function. If [Notify the bus error interrupt by function call] is selected in the GUI, the function having the specified name will be called when an interrupt occurs. Create the interrupt notification function as follows:
  void <*name of the interrupt notification function*> (void)
  For the interrupt notification function, note the contents of 5.21, Notes on Notification Functions.
- The status of bus error generation can be acquired by calling R_PG_ExtBus_ClearErrorFlags.

Example: A case where BusErrFunc has been specified as the name of the bus error interrupt notification function.

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

void func(void)
{
    R_PG_ExtBus_SetBus();       //Set up the bus pins and bus error monitoring.
}

//Bus error notification function
void BusErrFunc(void)
{
    bool addr_err;
    uint8_t master;
    uint16_t err_addr;

    //Aquire bus error status
    R_PG_ExtBus_GetErrorStatus(&addr_err, &master, &err_addr);
    if( addr_err ){
        //Processing when illegal address access error occurs
    }

    //Clear the bus error status registers
    R_PG_ExtBus_ClearErrorFlags();
}
```

5.5.2 R_PG_ExtBus_GetErrorStatus

Definition

bool R_PG_ExtBus_GetErrorStatus
(bool * addr_err, uint8_t * master, uint16_t * err_addr)

Description

Acquire the status of bus error generation

Parameter

bool * addr_err	The address of the storage area for the illegal address access error flag
uint8_t * master	The address of the storage area for ID code of bus master that accessed a bus when a bus error occurred ID code of bus master: 0:CPU 3: DTC
uint16_t * err_addr	The address of the storage area for upper 13 bits of an address that was accessed when a bus error occurred

Return value

true	Acquisition succeeded.
false	Acquisition failed.

File for output

R_PG_ExtBus.c

RPDL function

R_BSC_GetStatus

Details

- Acquires the status of bus error generation from the bus error status registers.
- Specify the address of storage area for an item to be acquired. Specify 0 for an item that is not required.

Example

Refer to the example of R_PG_ExtBus_SetBus

5.5.3 R_PG_ExtBus_ClearErrorFlags

Definition bool R_PG_ExtBus_ClearErrorFlags(void)

Description Clear the bus-error status registers

Parameter None

Return value

true	Clearing succeeded
false	Clearing failed

File for output R_PG_ExtBus.c

RPDL function R_BSC_Control

Details
- Clears the bus-error status registers (illegal address access error flag, ID code of bus master and a value of accessed address).
- The DMA interrupt request flag (IR flag) is cleared in this function.

Example Refer to the example of R_PG_ExtBus_SetBus

5.6 Data Transfer Controller (DTC)

5.6.1 R_PG_DTC_Set

Definition: bool R_PG_DTC_Set (void)

Description: Set the common options for DTC

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Dtc.c

RPDL function: R_DTC_Set

Details:

- This function configures the read skip control, address mode and the DTC vector table base address.

Example: A case where the setting is made as follows.

- The DTC vector table address has been set to 15000h.
- The transfer setting of which the transfer start trigger is IRQ0 has been made.

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

//DTC vector table
#pragma address dtc_vector_table = 0x00015000
uint32_t dtc_vector_table [256];

//Set up the DTC
void func(void)
{
    // Set the common options for DTC
    R_PG_DTC_Set();

    //Make the transfer setting of which the transfer start trigger is IRQ0
    R_PG_DTC_Set_IRQ0();

    //Make DTC be ready for the transfer start trigger
    R_PG_DTC_Activate();

    //Set up IRQ0
    R_PG_ExtInterrupt_Set_IRQ0();
}
```

5.6.2 R_PG_DTC_Set_<trigger source>

Definition

bool R_PG_DTC_Set_<trigger source> (void)

< trigger source > :
SWINT, CMT0 to 3, SPRI0, SPTI0, IRQ0 to 7, ADI0, S12ADI0 to 1, CMPI, TGIA0 to D7, TCIV4 and 7, TGIU5 to W5, GTCIA0 to C3, GTCIE0 to E3, GTCIV0 to V3, LOCOI, RXI0 to 2, TXI0 to 2, ICRXI0, ICTXI0

Description

Set the DTC transfer data

Parameter

None

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_Dtc.c

RPDL function

R_DTC_Create

Details

- Store the transfer data that will be triggered by transfer start trigger in specified address.
- The transfer data of the chain transfer will also be stored.
- If other transfer data has already been stored in the specified address, new data will be overwritten.
- This function does not set any interrupts used for transfer start triggers. Set up interrupts by each peripheral function.
- Select DTC as the request destination of interrupts used for the transfer start trigger.

Example

A case where the setting is made as follows.

- The DTC vector table address has been set to 15000h.
- The transfer setting of which the transfer start trigger is IRQ0 has been made.
- The transfer setting of which the transfer start trigger is IRQ1 has been made.

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

//DTC vector table
#pragma address dtc_vector_table = 0x00015000
uint32_t dtc_vector_table [256];

//Set up the DTC
void func(void)
{
    // Set the common options for DTC
    R_PG_DTC_Set();

    //Make the transfer setting of which the transfer start trigger is IRQ0
    R_PG_DTC_Set_IRQ0();

    //Make the transfer setting of which the transfer start trigger is IRQ1
    R_PG_DTC_Set_IRQ1();

    //Make DTC be ready for the transfer start trigger
    R_PG_DTC_Activate();

    //Set up IRQ0 and IRQ1
    R_PG_ExtInterrupt_Set_IRQ0();
    R_PG_ExtInterrupt_Set_IRQ1();
}
```

5.6.3 R_PG_DTC_Activate

Definition: bool R_PG_DTC_Activate (void)

Description: Make the DTC be ready for the transfer start trigger

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Dtc.c

RPDL function: R_DTC_Control

Details:

- Makes the DTC be ready for the transfer start trigger.
- Call R_PG_DTC_Set_<trigger source> to store the transfer data before calling this function.

Example: A case where the setting is made as follows.

- The DTC vector table address has been set to 15000h.
- The transfer setting of which the transfer start trigger is IRQ0 has been made.
- “Request is transferred to CPU when specified transfer is completed” has been selected in the interrupt setting.
- The chain transfer has been disabled.
- Irq0IntFunc has been specified as an IRQ0 interrupt notification function name.

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

//DTC vector table
#pragma address dtc_vector_table = 0x00015000
uint32_t dtc_vector_table [256];

//Set up the DTC
void func(void)
{
    // Set the common options for DTC
    R_PG_DTC_Set();

    //Make the transfer setting of which the transfer start trigger is IRQ0
    R_PG_DTC_Set_IRQ0();

    //Make DTC be ready for the transfer start trigger
    R_PG_DTC_Activate();

}

void Irq0IntFunc(void)
{
    //Disable the IRQ0
    //(After specified number of transfer completes, transfer will be executed
    // when the trigger is input. To stop the data transfer, disable the interrupt.)
    R_PG_ExtInterrupt_Disable_IRQ0();
}
```

5.6.4 R_PG_DTC_SuspendTransfer

Definition bool R_PG_DTC_SuspendTransfer (void)

Description Stop the data transfer

Parameter None

Return value

true	Stopping succeeded
false	Stopping failed

File for output R_PG_Dtc.c

RPDL function R_DTC_Control

Details

- Stops the data transfer.
- If transfer is stopped during data transfer, the accepted start request is active until the processing is completed.
  Call R_DTC_Activate to resume the transfer.

Example A case where the setting is made as follows.

- The DTC vector table address has been set to 15000h.
- The transfer setting of which the transfer start trigger is IRQ0 has been made.

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

//DTC vector table
#pragma address dtc_vector_table = 0x00015000
uint32_t dtc_vector_table [256];

//Set up the DTC
void func(void)
{
    // Set the common options for DTC
    R_PG_DTC_Set();

    //Make the transfer setting of which the transfer start trigger is IRQ0
    R_PG_DTC_Set_IRQ0();

    //Make DTC be ready for the transfer start trigger
    R_PG_DTC_Activate();

    //Set up IRQ0
    R_PG_ExtInterrupt_Set_IRQ0();
}

//Suspend the DTC transfer
void func2(void)
{
    R_PG_DTC_SuspendTransfer();
}

//Resume the DTC transfer
void func3(void)
{
    R_PG_DTC_Activate();
}
```

5.6.5 R_PG_DTC_GetTransmitStatus

Definition: bool R_PG_DTC_GetTransmitStatus (uint8_t * vector, bool * active)

Description: Get transfer status

Parameter

uint8_t * vector	The address of the storage area for the vector number of current data transfer (Valid when "* active" is 1)
bool * active	The address of the storage area for the progress flag. If this value is 1, the data transfer is processed.

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output: R_PG_Dtc.c

RPDL function: R_DTC_GetStatus

Details

- This function acquires the active flag and the vector number of the current data transfer.

Example

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

uint8_t vector;
bool active;

void func(void)
{
    //Get the DTC transfer status
    R_PG_DTC_GetTransmitStatus ( &vector, &active);

    if(active){
        switch( vector ){
            case 64:
                //Processing when the transfer of vector 64 is in progress
                break;
            case 65:
                //Processing when the transfer of vector 65 is in progress
                break;
            default:
        }
    }
}
```

5.6.6 R_PG_DTC_StopModule

Definition bool R_PG_DTC_StopModule (void)

Description Shut down the DTC

Parameter None

Return value

true	Shutting down succeeded
false	Shutting down failed

File for output R_PG_Dtc.c

RPDL function R_DTC_Destroy

Details
- This function shuts down the DTC and places it in the module-stop state.
- Disable the interrupt used for transfer start trigger before calling this function.

Example A case where the setting is made as follows.
- The DTC vector table address has been set to 15000h.
- The transfer setting of which the transfer start trigger is IRQ0 has been made.
- The transfer setting of which the transfer start trigger is IRQ1 has been made.

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

//DTC vector table
#pragma address dtc_vector_table = 0x00015000
uint32_t dtc_vector_table [256];

void func(void)
{
    // Set the common options for DTC
    R_PG_DTC_Set();

    //Make the transfer setting of which the transfer start trigger is IRQ0
    R_PG_DTC_Set_IRQ0();

    //Make the transfer setting of which the transfer start trigger is IRQ1
    R_PG_DTC_Set_IRQ1();

    //Make DTC be ready for the transfer start trigger
    R_PG_DTC_Activate();

    //Set up IRQ0 and IRQ1
    R_PG_ExtInterrupt_Set_IRQ0();
    R_PG_ExtInterrupt_Set_IRQ1();
}

void func2(void)
{
    //Disable IRQ0 and IRQ1
    R_PG_ExtInterrupt_Disable_IRQ0();
    R_PG_ExtInterrupt_Disable_IRQ1();
    //Shut down the DTC
    R_PG_DTC_StopModule();
}
```

5.7 I/O Ports

5.7.1 R_PG_IO_PORT_Set_P<*port number*>

Definition: bool R_PG_IO_PORT_Set_P<*port number*> (void)
<*port number*>: 1 to 9 and A, B, D, E and G

Description: Set up the I/O port

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_IO_PORT_P<*port number*>.c
<*port number*>: 1 to 9 and A, B, D, E and G

RPDL function: R_IO_PORT_Set

Details:

- Selects the direction (input or output) and input buffer for pins for which [Used as I/O port] was specified in the GUI.
- This function is used to set all pins in a port for which [Used as I/O port] has been selected.

Example:

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

void func(void)
{
      //Set P1.
      R_PG_IO_PORT_Set_P1();
}
```

5.7.2 R_PG_IO_PORT_Set_P<*port number*><*pin number*>

Definition

bool R_PG_IO_PORT_Set_P<*port number*><*pin number*> (void)

<*port number*>: 1 to 9 and A, B, D, E and G

<*pin number*>: 0 to 7

Description

Set up the I/O port pin

Parameter

None

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_IO_PORT_P<*port number*>.c

<*port number*>: 1 to 9 and A, B, D, E and G

RPDL function

R_IO_PORT_Set

Details

- Selects the direction (input or output), input buffer, pulling up, and open-drain output for a pin for which [Used as I/O port] was specified in the GUI.
- The setting only applies to one pin.

Example

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

void func(void)
{
    //Set P10.
    R_PG_IO_PORT_Set_P10();

    //Set P11.
    R_PG_IO_PORT_Set_P11();
}
```

5.7.3 R_PG_IO_PORT_Read_P<*port number*>

Definition

bool R_PG_IO_PORT_Read_P<*port number*> (uint8_t * data)

<*port number*>: 1 to 9 and A, B, D, E and G

Description

Read data from the I/O port register

Parameter

uint8_t * data	Destination for the storage of the read pin state

Return value

true	Reading proceeded correctly.
false	Reading failed.

File for output

R_PG_IO_PORT_P<*port number*>.c

<*port number*>: 1 to 9 and A, B, D, E and G

RPDL function

R_IO_PORT_Read

Details

- Reads an I/O port register to acquire the states of the pins.

Example

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

void func(void)
{
    uint8_t data

    //Acquire the states of P1 pins.
    R_PG_IO_PORT_Read_P1( &data );
}
```

5.7.4 R_PG_IO_PORT_Read_P<*port number*><*pin number*>

Definition

bool R_PG_IO_PORT_Read_P<*port number*><*pin number*> (uint8_t * data)

<*port number*>: 1 to 9 and A, B, D, E and G

<*pin number*>: 0 to 7

Description

Read 1-bit data from the I/O port register

Parameter

uint8_t * data	Destination for the storage of the read pin state

Return value

true	Reading proceeded correctly.
false	Reading failed.

File for output

R_PG_IO_PORT_P<*port number*>.c

(<*port number*>: 1 to 9 and A, B, D, E and G)

RPDL function

R_IO_PORT_Read

Details

- Reads an I/O port register to acquire the state of one pin.
- The value is stored in the lowest-order bit of *data.

Example

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

void func(void)
{
    uint8_t data_p10, data_p11;

    //Acquire the state of pin P10.
    R_PG_IO_PORT_Read_P10( & data_p10);

    //Acquire the state of pin P11.
    R_PG_IO_PORT_Read_P11( & data_p11);
}
```

5.7.5 R_PG_IO_PORT_Write_P<*port number*>

Definition

bool R_PG_IO_PORT_Write_P<*port number*> (uint8_t data)

<*port number*>: 1 to 9 and A, B, D, E and G

Description

Write data to the I/O port data register

Parameter

uint8_t data	Value to be written

Return value

true	Writing proceeded correctly.
false	Writing failed.

File for output

R_PG_IO_PORT_P<*port number*>.c

<*port number*>: 1 to 9 and A, B, D, E and G

RPDL function

R_IO_PORT_Write

Details

- Writes a value to an I/O port data register. A value written to the register is output from the output port.

Example

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

void func(void)
{
    //Set P1.
    R_PG_IO_PORT_Set_P1();

    //Output 0x03 from P1.
    R_PG_IO_PORT_Set_P1( 0x03 );
}
```

5.7.6 R_PG_IO_PORT_Write_P<*port number*><*pin number*>

Definition

bool R_PG_IO_PORT_Write_P<*port number*><*pin number*> (uint8_t data)

<*port number*>: 1 to 9 and A, B, D, E and G

<*pin number*>: 0 to 7

Description

Write 1-bit data to the I/O port data register

Parameter

uint8_t data	Value to be written

Return value

true	Writing proceeded correctly.
false	Writing failed.

File for output

R_PG_IO_PORT_P<*port number*>.c

<*port number*>: 1 to 9 and A, B, D, E and G

RPDL function

R_IO_PORT_Write

Details

- Writes a value to an I/O port data register. A value written to an output port is output. Store the value in the lowest-order bit of data.

Example

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

void func(void)
{
    //Set P10.
    R_PG_IO_PORT_Set_P10();

    //Set P11.
    R_PG_IO_PORT_Set_P11();

    //Output low level from P10.
    R_PG_IO_PORT_Write_P10( 0x00 );

    //Output high level from P11.
    R_PG_IO_PORT_Write_P11( 0x01 );
}
```

5.8 Multi-Function Timer Pulse Unit 3 (MTU3)

5.8.1 R_PG_Timer_Set_MTU_U<*unit number*>_<*channels*>

Definition: bool R_PG_Timer_Set_MTU_U<*unit number*>_<*channels*> (void)
<*unit number*>: 0
<*channel*>: C0 to C7 or C3_C4 or C6_C7

Description: Set up the MTU

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Timer_MTU_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 7

RPDL function: R_MTU3_Set, R_MTU3_Create

Details:

- Releases the MTU from the module-stop and makes initial settings.
- Interrupts of the MTU are set by this function. When the name of the interrupt notification function has been specified in the GUI, if an interrupt occurs in the CPU, the function having the specified name will be called. Create the interrupt notification function as follows:
  void <*name of the interrupt notification function*> (void)
  For the interrupt notification function, note the contents of 5.21, Notes on Notification Functions.
- If the interrupt propriety level is set to 0 in the GUI, a CPU interrupt does not occur. The state of a request flag can be acquired by calling R_PG_Timer_GetRequestFlag_MTU_U<*unit number*>_C<*channel number*>.
- When counting driven by an externally input clock, the external reset signal, input capture, or pulse output is in use, the direction (input or output) and input buffer for the pin to be used is set in this function.
- To start the count operation, call R_PG_Timer_StartCount_MTU_U<*unit number*>_C<*channel number*>(_<*phase*>) or R_PG_Timer_SynchronouslyStartCount_MTU_U<*unit number*> after calling this function.
- In complementary PWM mode or reset-synchronized PWM mode, paired channels are set up in the same time. Channels 3 and 4 are set up in channel 3 setting, channels 6 and 7 are set up in channel 6 setting.
- In complementary PWM mode or reset-synchronized PWM mode, PWM output is disabled in the initial state. To enable the pin output, call R_PG_Timer_ControlOutputPin_MTU_U<*unit number*>_<*channels*> before starting the count operation.

Example 1 A case where the setting is made as follows.

- MTU channel 6 was set up in normal mode
- Mtu6IcCmAIntFunc was specified as a compare match A interrupt notification function name

```
#include "R_PG_default.h" //Include "R_PG_<project name>.h" to use this function.

void func(void)
{
    R_PG_Timer_Set_MTU_U0_C6();       //Set up the MTU6
    R_PG_Timer_StartCount_MTU_U0_C6();       // Start the count operation
}

void Mtu6IcCmAIntFunc(void)
{
    //Processing in response to a compare match A interrupt
}
```

Example 2 A case where the setting is made as follows.

- MTU channel 3 and 4 were set up in complementary PWM mode

```
#include "R_PG_default.h" //Include "R_PG_<project name>.h" to use this function.

void func(void)
{
    //Set up the MTU3 and MTU4 in complementary PWM mode
    R_PG_Timer_Set_MTU_U0_C3_C4();

    //Enable PWM output pin 1 positive and negative phase
    R_PG_Timer_ControlOutputPin_MTU_U0_C3_C4(
        1, //p1 : enable
        1, //n1 : enable
        0, //p2 : disable
        0, //n2 : disable
        0, //p3 : disable
        0 //n3 : disable
    );

    // Start the MTU3 and 4 count operation
    R_PG_Timer_SynchronouslyStartCount_MTU_U0(
        0, //ch0
        0, //ch1
        0, //ch2
        1, //ch3
        1, //ch4
        0, //ch6
        0 //ch7
    );
}
```

5.8.2 R_PG_Timer_StartCount_MTU_U<*unit number*>_C<*channel number*>(_<*phase*>)

Definition

bool R_PG_Timer_StartCount_MTU_U<*unit number*>_C<*channel number*> (void)
<*unit number*>: 0
<*channel number*>: 0 to 7

bool R_PG_Timer_StartCount_MTU_U<*unit number*>_C<*channel number*>_<*phase*> (void)
<*unit number*>: 0
<*channel number*>: 5
<*phase*>: U, V or W

Description

Start the MTU count operation

Parameter

None

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_Timer_MTU_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 7

RPDL function

R_MTU3_ControlChannel

Details

- Starts the MTU count operation.
- Call R_PG_Timer_Set_MTU_U<*unit number*>_<*channels*> to make the initial settings before calling this function.
- In complementary PWM mode or reset-synchronized PWM mode, start the count operation of paired 2 channels simultaneously by R_PG_Timer_SynchronouslyStartCount_MTU_U<*unit number*>.
- R_PG_Timer_StartCount_MTU_U0_C5 can start the count of U, V, and W phase simultaneously.

Example

A case where the setting is made as follows.

- MTU channel 1 was set up
- Mtu1IcCmAIntFunc was specified as the compare match A interrupt notification function name

```
#include "R_PG_default.h" //Include "R_PG_<project name>.h" to use this function.

void func(void)
{
    R_PG_Timer_Set_MTU_U0_C1();       //Set up the MTU1
    R_PG_Timer_StartCount_MTU_U0_C1();       // Start the count operation
}

void Mtu1IcCmAIntFunc(void)
{
    R_PG_Timer_HaltCount_MTU_U0_C1();       //Halt the count operation
    func_cmA();       //Processing in response to a compare match A interrupt
    R_PG_Timer_StartCount_MTU_U0_C1();       //Resume the count operation
}
```

5.8.3 R_PG_Timer_SynchronouslyStartCount_MTU_U<*unit number*>

Definition

bool R_PG_Timer_SynchronouslyStartCount_MTU_U<*unit number*>
(bool ch0, bool ch1, bool ch2, bool ch3, bool ch4, bool ch6, bool ch7)
<*unit number*>: 0

Description

Start the MTU count operation of two or more channels simultaneously

Parameter

bool ch0	Count operation of channel 0 (0:Do not start count 1:Start count)
bool ch1	Count operation of channel 1 (0:Do not start count 1:Start count)
bool ch2	Count operation of channel 2 (0:Do not start count 1:Start count)
bool ch3	Count operation of channel 3 (0:Do not start count 1:Start count)
bool ch4	Count operation of channel 4 (0:Do not start count 1:Start count)
bool ch6	Count operation of channel 6 (0:Do not start count 1:Start count)
bool ch7	Count operation of channel 7 (0:Do not start count 1:Start count)

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_Timer_MTU_U<*unit number*>.c
<*unit number*>: 0

RPDL function

R_MTU3_ControlUnit

Details

- Starts the MTU count operation of two or more channels simultaneously.
- Call R_PG_Timer_Set_MTU_U<*unit number*>_<*channels*> to make the initial settings before calling this function.
- In complementary PWM mode or reset-synchronized PWM mode, start the count operation of paired 2 channels simultaneously by this function.

Example

Refer to the example 2 of R_PG_Timer_Set_MTU_U<*unit number*>_<*channels*>

5.8.4 R_PG_Timer_HaltCount_MTU_U<*unit number*>_C<*channel number*>(_<*phase*>)

Definition

bool R_PG_Timer_HaltCount_MTU_U<*unit number*>_C<*channel number*> (void)
<*unit number*>: 0
<*channel number*>: 0 to 7

bool R_PG_Timer_HaltCount_MTU_U<*unit number*>_C<*channel number*>_<*phase*> (void)
<*unit number*>: 0
<*channel number*>: 5
<*phase*>: U, V or W

Description

Halt the MTU count operation

Parameter

None

Return value

true	Halting succeeded.
false	Halting failed.

File for output

R_PG_Timer_MTU_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 7

RPDL function

R_MTU3_ControlChannel

Details

- Halts the MTU count operation.
- To make the MTU resume counting, call R_PG_Timer_StartCount_MTU_U<*unit number*>_C<*channel number*>(_<*phase*>) or R_PG_Timer_SynchronouslyStartCount_MTU_U<*unit number*>.
- R_PG_Timer_HaltCount_MTU_U0_C5 can stop the count of U, V, and W phase simultaneously.

Example

A case where the setting is made as follows.

- MTU channel 1 was set up
- Mtu1IcCmAIntFunc was specified as the compare match A interrupt notification function name

```
#include "R_PG_default.h" //Include "R_PG_<project name>.h" to use this function.

void func(void)
{
    R_PG_Timer_Set_MTU_U0_C1();       //Set up the MTU1
    R_PG_Timer_StartCount_MTU_U0_C1();       // Start the count operation
}

void Mtu1IcCmAIntFunc(void)
{
    R_PG_Timer_HaltCount_MTU_U0_C1();       //Halt the count operation
    func_cmA();       //Processing in response to a compare match A interrupt
    R_PG_Timer_StartCount_MTU_U0_C1();       //Resume the count operation
}
```

5.8.5 R_PG_Timer_GetCounterValue_MTU_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_GetCounterValue_MTU_U<*unit number*>_C<*channel number*>
(uint16_t * counter_val)
<*unit number*>: 0
<*channel number*>: 0 to 4 and 6, 7

bool R_PG_Timer_GetCounterValue_MTU_U<*unit number*>_C<*channel number*>
(uint16_t * counter_u_val, uint16_t * counter_v_val, uint16_t * counter_w_val)
<*unit number*>: 0
<*channel number*>: 5

Description

Acquire the MTU counter value

Parameter

For MTU0 to MTU4 and MTU6, MTU7

uint16_t * counter_val	Destination for the storage of the counter value

For MTU5

uint16_t * counter_u_val	Destination for the storage of the counter U value
uint16_t * counter_v_val	Destination for the storage of the counter V value
uint16_t * counter_w_val	Destination for the storage of the counter value

Return value

true	Acquisition of the counter value succeeded.
false	Acquisition of the counter value failed.

File for output

R_PG_Timer_MTU_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 7

RPDL function

R_MTU3_ReadChannel

Details

- Acquires the counter value of a MTU.

Example

A case where the setting is made as follows.

- MTU channel 0 was set up
- Set TGRA as an input capture register and enable an input capture A interrupt
- Mtu0IcCmAIntFunc was specified as the input capture A interrupt notification function name

```
#include "R_PG_default.h" //Include "R_PG_<project name>.h" to use this function.

uint16_t counter_val;

void func(void)
{
      R_PG_Timer_Set_MTU_U0_C0();       //Set up the MTU0
      R_PG_Timer_StartCount_MTU_U0_C0();       // Start the count operation
}

void Mtu0IcCmAIntFunc(void)
{
      // Acquire the value of the MTU0 counter
      R_PG_Timer_GetCounterValue_MTU_U0_C0( & counter_val );
}
```

5.8.6 R_PG_Timer_SetCounterValue_MTU_U<*unit number*>_C<*channel number*>(_<*phase*>)

Definition

bool R_PG_Timer_SetCounterValue_MTU_U<*unit number*>_C<*channel number*>
(uint16_t counter_val)

<*unit number*>: 0 <*channel number*>: 0 to 4 and 6, 7

bool R_PG_Timer_SetCounterValue_MTU_U<*unit number*>_C<*channel number*>_<*phase*>
(uint16_t counter_val)

<*unit number*>: 0 <*channel number*>: 5 <*phase*>: U, V or W

bool R_PG_Timer_SetCounterValue_MTU_U<*unit number*>_C<*channel number*>
(uint16_t counter_u_val, uint16_t counter_v_val, uint16_t counter_w_val)

<*unit number*>: 0 <*channel number*>: 5

Description

Set the MTU counter value

Parameter

For MTU0 to MTU7

uint16_t counter_val	Value to be written to the counter

For MTU5

uint16_t counter_u_val	Value to be written to the counter U
uint16_t counter_v_val	Value to be written to the counter V
uint16_t counter_w_val	Value to be written to the counter W

Return value

true	Setting of the counter value succeeded.
false	Setting of the counter value failed.

File for output

R_PG_Timer_MTU_U<*unit number*>_C<*channel number*>.c

<*unit number*>: 0

<*channel number*>: 0 to 7

RPDL function

R_MTU3_ControlChannel

Details

- Set the counter value of a MTU.

Example

A case where the setting is made as follows.

- MTU channel 1 was set up
- Set TGRA as an output compare register and enable a compare match A interrupt
- Mtu1IcCmAIntFunc was specified as the compare match A interrupt notification function name

```
#include "R_PG_default.h" //Include "R_PG_<project name>.h" to use this function.

void func (void)
{
      R_PG_Timer_Set_MTU_U0_C1();    //Set up the MTU1
      R_PG_Timer_StartCount_MTU_U0_C1();    // Start the count operation
}

void Mtu1IcCmAIntFunc(void)
{
      R_PG_Timer_SetCounterValue_MTU_U0_C1( 0 );    //Clear the counter
}
```

5.8.7 R_PG_Timer_GetRequestFlag_MTU_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_GetRequestFlag_MTU_U<*unit number*>_C<*channel number*>
(bool* cm_ic_a, bool* cm_ic_b, bool* cm_ic_c, bool* cm_ic_d,
bool* cm_e, bool* cm_f, bool* ov, bool* un);
<*unit number*>: 0
<*channel number*>: 0 to 4 and 6, 7

bool R_PG_Timer_GetRequestFlag_MTU_U<*unit number*>_C<*channel number*>
(bool* cm_ic_u, bool* cm_ic_v, bool* cm_ic_w);
<*unit number*>: 0
<*channel number*>: 5

Description

Acquire and clear the MTU interrupt flags

Parameter

bool* cm_ic_a	The address of the storage area for the compare match/input capture A flag
bool* cm_ic_b	The address of the storage area for the compare match/input capture B flag
bool* cm_ic_c	The address of the storage area for the compare match/input capture C flag
bool* cm_ic_d	The address of the storage area for the compare match/input capture D flag
bool* cm_e	The address of the storage area for the compare match E flag
bool* cm_f	The address of the storage area for the compare match F flag
bool* ov	The address of the storage area for the overflow flag
bool* un	The address of the storage area for the underflow flag
bool* cm_ic_u	The address of the storage area for the compare match/input capture U flag
bool* cm_ic_v	The address of the storage area for the compare match/input capture V flag
bool* cm_ic_w	The address of the storage area for the compare match/input capture W flag

Available flags for each channel are as follows.

MTU0	cm_ic_a to cm_ic_d, cm_e, cm_f, and ov
MTU1, 2	cm_ic_a, cm_ic_b, ov, and un
MTU3, 4, 6, 7	cm_ic_a to cm_ic_d, and ov
MTU5	cm_ic_u, cm_ic_v, and cm_ic_w
MTU3, 6 (complementary PWM mode and reset-synchronized PWM mode)	cm_ic_a and cm_ic_b
MTU4, 7 (complementary PWM mode and reset-synchronized PWM mode)	cm_ic_a, cm_ic_b, and un

Return value

true	Acquisition of the flags succeeded
false	Acquisition of the flags failed

File for output

R_PG_Timer_MTU_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 7

RPDL function

R_MTU3_ReadChannel

Details

- This function acquires the interrupt flags of MTU.
- All flags will be cleared in this function.
- Specify the address of storage area for the flags to be acquired.
  Specify 0 for a flag that is not required.

Example A case where the setting is made as follows.

- MTU channel 1 was set up
- TGRA is set as an output compare register and the compare match interrupt is enabled
- The priority level of compare match interrupt is set to 0

```
#include "R_PG_default.h" //Include "R_PG_<project name>.h" to use this function.

bool cma_flag;

void func(void)
{
    R_PG_Timer_Set_MTU_U0_C1();    //Set up the MTU1
    R_PG_Timer_StartCount_MTU_U0_C1();    // Start the count operation

    //Wait for the compare match A
    do{
        R_PG_Timer_GetRequestFlag_MTU_U0_C1(
            & cma_flag, //a
            0, //b
            0, //c
            0, //d
            0, //e
            0, //f
            0, //e
            0, //ov
            0 //un
        );
    } while( !cma_flag );

    //Processing in response to a compare match A

}
```

5.8.8 R_PG_Timer_StopModule_MTU_U<*unit number*>

Definition

bool R_PG_Timer_StopModule_MTU_U<*unit number*> (void)

<*unit number*>: 0

Description

Shut down the MTU unit

Parameter

None

Return value

true	Shutting down succeeded
false	Shutting down failed

File for output

R_PG_Timer_MTU_U<*unit number*>.c

<*unit number*>: 0

RPDL function

R_MTU3_Destroy

Details

- Stops a MTU and places it in the module-stop state. If two or more channels are running when this function is called, all channels will be stopped. Call R_PG_Timer_HaltCount_MTU_U<*unit number*>_C<*channel number*>(_<*phase*>) to stop a single channel.

Example

A case where the setting is made as follows.

- MTU channel 1 was set up
- Set TGRA as an output compare register and enable a compare match A interrupt Mtu1IcCmAIntFunc was specified as the compare match A interrupt notification function name

```
#include "R_PG_default.h" //Include "R_PG_<project name>.h" to use this function.

void func(void)
{
    R_PG_Timer_Set_MTU_U0_C1();    //Set up the MTU1
    R_PG_Timer_StartCount_MTU_U0_C1();    // Start the count operation
}

void Mtu1IcCmAIntFunc(void)
{
    // Stop the MTU unit 0
    R_PG_Timer_StopModule_MTU_U0();
}
```

5.8.9 R_PG_Timer_GetTGR_MTU_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_GetRequestFlag_MTU_U<*unit number*>_C<*channel number*>
(uint16_t* tgr_a_val, uint16_t* tgr_b_val, uint16_t* tgr_c_val,
uint16_t* tgr_d_val, uint16_t* tgr_e_val, uint16_t* tgr_f_val);
<*unit number*>: 0
<*channel number*>: 0 to 4 or 6, 7

bool R_PG_Timer_GetRequestFlag_MTU_U<*unit number*>_C<*channel number*>
(uint16_t * tgr_u_val, uint16_t * tgr_v_val, uint16_t * tgr_w_val);
<*unit number*>: 0
<*channel number*>: 5

Description

Acquire the general register value

Parameter

uint16_t* tgr_a_val	The address of the storage area for the general register A value
uint16_t* tgr_b_val	The address of the storage area for the general register B value
uint16_t* tgr_c_val	The address of the storage area for the general register C value
uint16_t* tgr_d_val	The address of the storage area for the general register D value
uint16_t* tgr_e_val	The address of the storage area for the general register E value
uint16_t* tgr_f_val	The address of the storage area for the general register F value
uint16_t* tgr_u_val	The address of the storage area for the general register U value
uint16_t* tgr_v_val	The address of the storage area for the general register V value
uint16_t* tgr_w_val	The address of the storage area for the general register W value

Available arguments for each channel are as follows.

MTU0	tgr_a_val to tgr_f_val
MTU1, 2	tgr_a_val and tgr_b_val
MTU3, 4, 6, 7	tgr_a_val to tgr_d_val
MTU5	tgr_u_val to tgr_w_val
MTU3, 6 (complementary PWM mode)	tgr_a_val to tgr_e_val
MTU4, 7 (complementary PWM mode)	tgr_a_val to tgr_f_val
MTU3, 4, 6, 7 (reset-synchronized PWM mode)	tgr_a_val to tgr_d_val

Return value

true	Acquisition of the flags succeeded
false	Acquisition of the flags failed

File for output

R_PG_Timer_MTU_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 7

RPDL function

R_MTU3_ReadChannel

Details

- This function acquires the general register value.
- Specify the address of storage area for an item to be acquired. Specify 0 for an item that is not required.

Example

A case where the setting is made as follows.

- MTU channel 0 was set up
- Set TGRA as an input capture register and enable an input capture A interrupt
- Mtu0IcCmAIntFunc was specified as the input capture A interrupt notification function name

```
#include "R_PG_default.h" //Include "R_PG_<project name>.h" to use this function.

uint16_t tgr_a_val;

void func(void)
{
	R_PG_Timer_Set_MTU_U0_C0();		//Set up the MTU0
	R_PG_Timer_StartCount_MTU_U0_C0();		// Start the count operation
}

void Mtu0IcCmAIntFunc(void)
{
	// Acquire the value of the TGRA
	R_PG_Timer_GetTGR_MTU_U0_C0(
		& tgr_a_val, //a
		0, //b
		0, //c
		0, //d
		0, //e
		0 //f
	);
}
```

5.8.10 R_PG_Timer_SetTGR_<*general register*>_MTU_U<*unit number*>_C<*channel number*>

Definition bool R_PG_Timer_SetTGR_<*general register*>_MTU_U<*unit number*>_C<*channel number*> (uint16_t value);

<*general register*>:

MTU1, 2	: A or B
MTU3, 4, 6, 7	: A, B, C or D
MTU5	: U, V or W
MTU3, 4, 6, 7 (complementary PWM mode)	: A, B, C, D, E(*1), or F(*1)
MTU3, 4, 6, 7 (reset-synchronized PWM mode)	: A, B, C(*2), or D(*3)

(*1 Only when the double buffer operation is enabled)
(*2 Only when the TGRC is used as a buffer register)
(*3 Only when the TGRD is used as a buffer register)

<*unit number*>: 0

<*channel number*>: 0 to 7

Description Set the general register value

Parameter

uint16_t value	Value to be written to the general register

Return value

true	Setting of the general register succeeded.
false	Setting of the general register failed.

File for output R_PG_Timer_MTU_U<*unit number*>_C<*channel number*>.c

<*unit number*>: 0

<*channel number*>: 0 to 7

RPDL function R_MTU3_ControlChannel

Details

- This function sets the general register value.

Example

A case where the setting is made as follows.

- MTU channel 1 was set up
- Set TGRA as an output compare register and enable a compare match A interrupt
- Mtu1IcCmAIntFunc was specified as the compare match A interrupt notification function name

```
#include "R_PG_default.h" //Include "R_PG_<project name>.h" to use this function.

void func (void)
{
    R_PG_Timer_Set_MTU_U0_C1();    //Set up the MTU1
    R_PG_Timer_StartCount_MTU_U0_C1();    // Start the count operation
}

void Mtu1IcCmAIntFunc(void)
{
    R_PG_Timer_SetTGR_A__MTU_U0_C1( 1000 );    //Set TGRA
}
```

5.8.11 R_PG_Timer_SetBuffer_AD_MTU_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_SetBuffer_AD_MTU_U<*unit number*>_C<*channel number*>
(uint16_t tadcobr_a_val, uint16_t tadcobr_b_val);
<*unit number*>: 0
<*channel number*>: 4 or 7

Description

Set A/D converter start request cycle set buffer registers (TADCOBRA and TADCOBRB)

Conditions for output

The buffer transfer of A/D converter start request cycle value is enabled.

Parameter

uint16_t tadcobr_a_val	Value to be written to TADCOBRA
uint16_t tadcobr_b_val	Value to be written to TADCOBRB

Return value

true	Setting of the counter value succeeded.
false	Setting of the counter value failed.

File for output

R_PG_Timer_MTU_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 3(*), 4, 6(*), 7
(* complementary PWM mode and reset-synchronized PWM mode)

RPDL function

R_MTU3_ControlChannel

Details

- This function sets the TADCOBRA and TADCOBRB values.

Example

A case where the setting is made as follows.

- Buffer transfer of A/D converter start request cycle set register has been enabled

```
#include "R_PG_default.h" //Include "R_PG_<project name>.h" to use this function.

void func (void)
{
      R_PG_Timer_Set_MTU_U0_C4();    //Set up the MTU1
      R_PG_Timer_StartCount_MTU_U0_C4();    // Start the count operation
}

void Mtu1IcCmAIntFunc(void)
{
      // Set TADCOBRA and TADCOBRB
      R_PG_Timer_SetBuffer_AD_MTU_U0_C4( 0x10, 0x20 );
}
```

5.8.12 R_PG_Timer_SetBuffer_CycleData_MTU_U<*unit number*>_<*channels*>

Definition

bool R_PG_Timer_SetBuffer_CycleData_MTU_U<*unit number*>_<*channels*>
(uint16_t tcbr_val);
<*unit number*>: 0
<*channels*>: C3_C4 or C6_C7

Description

Set the cycle buffer register

Conditions for output

MTU channels are set to complementary PWM mode

Parameter

uint16_t tcbr_val	Value to be written to the cycle buffer registerr

Return value

true	Setting of the counter value succeeded.
false	Setting of the counter value failed.

File for output

R_PG_Timer_MTU_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 3 or 6

RPDL function

R_MTU3_ControlUnit

Details

- This function sets the cycle buffer register (TCBRA (channel 3 and 4) or TCBRB (channel 6 and 7).

Example

```
#include "R_PG_default.h" //Include "R_PG_<project name>.h" to use this function.

void func (void)
{
      R_PG_Timer_SetBuffer_CycleData_MTU_U0_C3_C4(0x1000);
}
```

5.8.13 R_PG_Timer_SetOutputPhaseSwitch_MTU_U<*unit number*>_<*channels*>

Definition

bool R_PG_Timer_SetOutputPhaseSwitch_MTU_U<*unit number*>_<*channels*>
(uint8_t output_level);
<*unit number*>: 0
<*channels*>: C3_C4

Description

Switch PWM output level

Conditions for output

- The MTU channels are set to complementary PWM mode or reset-synchronized PWM mode
- The brushless DC motor control is enabled and the software is selected for the output control method

Parameter

uint8_t output_level	PWM output setting (0 to 7)

The output level for each value is as follows

Value	MTIOC3B U phase	MTIOC4A V phase	MTIOC4B W phase	MTIOC3D U phase	MTIOC4C V phase	MTIOC4D W phase
0	OFF	OFF	OFF	OFF	OFF	OFF
1	ON	OFF	OFF	OFF	OFF	ON
2	OFF	ON	OFF	ON	OFF	OFF
3	OFF	ON	OFF	OFF	OFF	ON
4	OFF	OFF	ON	OFF	ON	OFF
5	ON	OFF	OFF	OFF	ON	OFF
6	OFF	OFF	ON	ON	OFF	OFF
7	OFF	OFF	OFF	OFF	OFF	OFF

Return value

true	Setting of the counter value succeeded.
false	Setting of the counter value failed.

File for output

R_PG_Timer_MTU_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 3

RPDL function

R_MTU3_ControlUnit

Details

- This function switches the PWM output level in brushless DC motor control

Example

```
#include "R_PG_default.h" //Include "R_PG_<project name>.h" to use this function.

void func (void)
{
      R_PG_Timer_SetOutputPhaseSwitch_MTU_U0_C3_C4 (0x7);
}
```

5.8.14 R_PG_Timer_ControlOutputPin_MTU_U<*unit number*>_<*channels*>

Definition

bool R_PG_Timer_ControlOutputPin_MTU_U<*unit number*>_<*channels*>
(bool p1_enable, bool n1_enable, bool p2_enable, bool n2_enable,
bool p3_enable, bool n3_enable)
<*unit number*>: 0
<*channels*>: C3_C4 or C6_C7

Description

Enable or disable the PWM output

Conditions for output

MTU channels are set to complementary PWM mode or reset-synchronized PWM mode

Parameter

bool p1_enable	U positive phase (MTIOCmB) output (0: Disable 1: Enable)
bool n1_enable	U negative phase (MTIOCmD) output (0: Disable 1: Enable)
bool p2_enable	V positive phase (MTIOCnA) output (0: Disable 1: Enable)
bool n2_enable	V negative phase (MTIOCnC) output (0: Disable 1: Enable)
bool p3_enable	W positive phase (MTIOCnB) output (0: Disable 1: Enable)
bool n3_enable	W negative phase (MTIOCnD) output (0: Disable 1: Enable)

m : 3, 6 n : 4, 7

Return value

true	Setting of the counter value succeeded.
false	Setting of the counter value failed.

File for output

R_PG_Timer_MTU_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 3 or 6

RPDL function

R_MTU3_ControlUnit

Details

- This function enables or disables PWM output in complementary PWM mode or reset-synchronized PWM mode.
- In complementary PWM mode or reset-synchronized PWM mode, PWM output is disabled in the initial state. To enable the pin output, call this function before starting the count operation.

Example

Refer to the example 2 of R_PG_Timer_Set_MTU_U<*unit number*>_<*channels*>

5.8.15 R_PG_Timer_SetBuffer_PWMOutputLevel_MTU_U<*unit number*>_<*channels*>

Definition

bool R_PG_Timer_SetBuffer_PWMOutputLevel_MTU_U<*unit number*>_<*channels*>
(bool p1_high, bool n1_high, bool p2_high, bool n2_high,
bool p3_high, bool n3_high)
<*unit number*>: 0
<*channels*>: C3_C4 or C6_C7

Description

Set the PWM output level in the buffer register

Conditions for output

MTU channels are set to complementary PWM mode or reset-synchronized PWM mode

Parameter

bool p1_high	U positive phase (MTIOCmB) output
bool n1_high	U negative phase (MTIOCmD) output
bool p2_high	V positive phase (MTIOCnA) output
bool n2_high	V negative phase (MTIOCnC) output
bool p3_high	W positive phase (MTIOCnB) output
bool n3_high	W negative phase (MTIOCnD) output

m : 3, 6 n : 4, 7

The output level in each value is as follows

Value	Category	Positive phase	Negative phase
0	Active level	Low	Low
	Initial output	Low	Low
	Compare match when up count	Low	High
	Compare match when down count	High	Low
1	Active level	High	High
	Initial output	High	High
	Compare match when up count	High	Low
	Compare match when down count	Low	High

Return value

true	Setting of the counter value succeeded.
false	Setting of the counter value failed.

File for output

R_PG_Timer_MTU_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 3 or 6

RPDL function

R_MTU3_ControlUnit

Details

- This function sets the output level settings to the timer output level buffer register (TOLBRA (channel 3 and 4) or TOLBRB (channel 6 and 7)

Example

```
#include "R_PG_default.h" //Include "R_PG_<project name>.h" to use this function.
void func (void)
{
    R_PG_Timer_SetBuffer_PWMOutputLevel_MTU_U0_C3_C4( 0, 0, 0, 0, 0, 0 );
}
```

5.8.16 R_PG_Timer_ControlBufferTransfer_MTU_U<*unit number*>_<*channels*>

Definition

bool R_PG_Timer_ControlBufferTransfer_MTU_U<*unit number*>_<*channels*>
(bool enable)
<*unit number*>: 0
<*channels*>: C3_C4 or C6_C7

Description

Enable or disable buffer transfer from the buffer registers to the temporary registers

Conditions for output

- The MTU channels are set to complementary PWM mode
- Interrupt skipping function 1 is selected for the interrupt skipping mode

Parameter

bool enable	Buffer transfer control (0 :Disable 1 :Enable)

Return value

true	Setting of the counter value succeeded.
false	Setting of the counter value failed.

File for output

R_PG_Timer_MTU_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 3 or 6

RPDL function

R_MTU3_ControlUnit

Details

- This function enables or disables transfer from the buffer registers used in complementary PWM mode to the temporary registers.

Example

```
#include "R_PG_default.h" //Include "R_PG_<project name>.h" to use this function.

void func (void)
{
    R_PG_Timer_ControlBufferTransfer_MTU_U0_C3_C4( 1 );
}
```

5.9 Port Output Enable 3 (POE3)

5.9.1 R_PG_POE_Set

Definition bool R_PG_POE_Set (void)

Description Set up the POE

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_POE.c

RPDL function R_POE_Set, R_POE_Create

- Sets up the output control of MTU0, 3, 4, 6, 7 and GPT0, 1, 2, 3 pins, the POE pins used for high-impedance request signal input, and the output enable interrupt.
- The MTU and GPT module is not set up in this function.
- Do not set pins that are not used for MTU output.
- When the name of the interrupt notification function has been specified in the GUI, if an interrupt occurs in the CPU, the function having the specified name will be called. Create the interrupt notification function as follows:
  void <*name of the interrupt notification function*> (void)
  For the interrupt notification function, note the contents of 5.21, Notes on Notification Functions.

Example

A case where the setting is made as follows.

- The output enable interrupt 2(OEI2) has been set
  PoeOei2IntFunc has been specified as an interrupt notification function name

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
      R_PG_POE_Set();       // Set up the POE
}

void PoeOei2IntFunc (void)
{
      // Processing when the output enable interrupt occurs
}
```

5.9.2 R_PG_POE_SetHiZ_<*Timer channels*>

Definition bool R_PG_POE_SetHiZ_<*Timer channels*>(void)

<*Timer channels*>: MTU3_4, MTU6_7, MTU0, GPT0_1, GPT2_3

Description Place the timer output pins in high-impedance state

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_POE.c

RPDL function R_POE_Control

Details Places MTU0, 3, 4, 6, 7, or GPT0, 1, 2, 3 output pins in high-impedance state.

Example A case where the setting is made as follows.

- MTU0 pin output has been set (Setting of MTU)
- MTU0 output pins have been set to be controlled by the high impedance request

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func1(void)
{
    R_PG_Timer_Set_MTU_U0_C0();    //Set up the MTU0
    R_PG_POE_Set();    // Set up the POE
    R_PG_Timer_StartCount_MTU_U0_C0();    //Start the count operation of MTU0
}

void func2(void)
{
    R_PG_POE_SetHiZ_MTU0();    // Place the MTU0 output pins in high-impedance state
}
```

5.9.3 R_PG_POE_GetRequestFlagHiZ_<*Timer channels*>

Definition

bool R_PG_POE_GetRequestFlagHiZ_MTU3_4 (bool* poe0)
bool R_PG_POE_GetRequestFlagHiZ_MTU6_7 (bool* poe4)
bool R_PG_POE_GetRequestFlagHiZ_MTU0 (bool* poe8)
bool R_PG_POE_GetRequestFlagHiZ_GPT0_1 (bool* poe10)
bool R_PG_POE_GetRequestFlagHiZ_GPT2_3 (bool* poe11)

Description

Acquire the high-impedance request flags

Parameter

bool* poe0	The address of the storage area for POE0# high-impedance request flags
bool* poe4	The address of the storage area for POE4# high-impedance request flags
bool* poe8	The address of the storage area for POE8# high-impedance request flags
bool* poe10	The address of the storage area for POE10# high-impedance request flags
bool* poe11	The address of the storage area for POE11# high-impedance request flags

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output

R_PG_POE.c

RPDL function

R_POE_GetStatus

Details

- Acquires the flags of high-impedance request signals input to POEn#pins(n:0 to 9) (POEnF n:0 to 9).
- Specify the address of storage area for the flags to be acquired. Specify 0 for a flag that is not required.
- The flag is valid only when the POE pin is set to a high-impedance request input in GUI.

Example

A case where the setting is made as follows.

- MTU3 and 4 pin output has been set (Setting of MTU)
- MTU3 and 4 output pins have been set to be controlled by the high impedance request
- POE0 has been selected as a high-impedance request signal input

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

bool poe0;

void func(void)
{
    R_PG_Timer_Set_MTU_U0_C3();     //Set up the MTU
    R_PG_POE_Set();     // Set up the POE
    R_PG_Timer_StartCount_MTU_U0_C3();     //Start the count operation of MTU

    //Wait for the high-impedance request signal to be input
    do{
        R_PG_POE_GetRequestFlagHiZ_MTU3_4( &poe0 );
    }while( ! poe0 );

    //Processing when the high-impedance request signal is input

    R_PG_POE_ClearFlag_MTU3_4();     //Clear high-impedance request flag
}
```

5.9.4 R_PG_POE_GetShortFlag_<*Timer channels*>

Definition bool R_PG_POE_GetShortFlag_MTU3_4 (bool * detected)
bool R_PG_POE_GetShortFlag_MTU6_7 (bool * detected)

Description Acquire the MTU output short flags

Parameter

bool* detected	The address of the storage area for the output short flag (MTU3,4:OSF1 or MTU6,7:OSF2)

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output R_PG_POE.c

RPDL function R_POE_GetStatus

Details
- Acquires the MTU3 ,4 or MTU6,7 complementary PWM output short flags (MTU3,4:OSF1 or MTU6,7:OSF2).

Example A case where the setting is made as follows.
- The output enable interrupt1(OEI1) has been set.
- PoeOei1IntFunc has been specified as the output enable interrupt 1 notification function name.

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
    R_PG_POE_Set();    // Set up the POE
}

void PoeOei1IntFunc(void)
{
    bool detected;

    //Acquire the output short flag
    R_PG_POE_GetShortFlag_MTU3_4 (&detected);

    if( detected ){
        //Processing when MTU3,4 output short is detected

        R_PG_POE_ClearFlag_MTU3_4();    // Clear the output short flag(OSF1)
    }
}
```

5.9.5 R_PG_POE_ClearFlag_<*Timer channels*>

Definition bool R_PG_POE_ClearFlag_<*Timer channels*> (void)

<*Timer channels*>: MTU3_4, MTU6_7, MTU0, GPT0_1, GPT2_3

Description Clear the high-impedance request flags and the output short flags

Parameter None

Return value

true	Clearing succeeded
false	Clearing failed

File for output R_PG_POE.c

RPDL function R_POE_Control

Details

- Clears the high-impedance request flags and the output short flags.
- The flags that shall be cleared by each function are as follows.

Timer channels	Flags
MTU3, 4	POE0 request flag (POE0F), MTU3,4 output short flag(OSF1)
MTU6, 7	POE4 request flag (POE4F), MTU6,7 output short flag(OSF2)
MTU0	POE8 request flag (POE8F)
GPT0,1	POE10 request flag (POE10F)
GPT2,3	POE11 request flag (POE11F)

Example Refer to the example of R_PG_POE_GetShortFlag_<*Timer channels*>

5.10 General PWM Timer (GPT)

5.10.1 R_PG_Timer_Set_GPT_U<*unit number*>

Definition: bool R_PG_Timer_Set_GPT_U<*unit number*> (void)
<*unit number*>: 0

Description: Set up the GPT

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Timer_GPT_U<*unit number*>.c
<*unit number*>: 0

RPDL function: R_GPT_Set, R_GPT_ControlUnit

Details:

- Releases the GPT from the module-stop and sets the timer input/output pins to be used. This function also sets up LOCO count function when it has been set in GUI. Call this function before calling R_PG_Timer_Set_GPT_U<*unit number*>_C<*channel number*>.
- To start the LOCO count, call R_PG_Timer_StartCount_LOCO_GPT_U<*unit number*> after calling this function.

Example:

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
      R_PG_Timer_Set_GPT_U0();    // Release GPT form module stop
      R_PG_Timer_Set_GPT_U0_C0();    // Set up GPT0
      R_PG_Timer_SetGTCCR_A_GPT_U0_C0( 0x6000 );    // Set GTCCRA
      R_PG_Timer_SetGTCCR_C_GPT_U0_C0( 0x4000 );    // Set GTCCRC
      R_PG_Timer_StartCount_GPT_U0_C0();    // Start the count operaion
}
```

5.10.2 R_PG_Timer_Set_GPT_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_Set_GPT_U<*unit number*>_C<*channel number*> (void)
<*unit number*>: 0
<*channel number*>: 0 to 3

Description

Set up the GPT channel

Parameter

None

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_Timer_GPT_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 3

RPDL function

R_GPT_Create, R_GPT_ControlChannel

Details

- This function makes initial settings for GPT channel.
- Call R_PG_Timer_Set_GPT_U<*unit number*> before calling this function to release GPT from module-stop state.
- The compare capture registers (GTCCRA to GTCCRF) are not set in this function. Call functions R_PG_Timer_SetGTCCR_n_GPT_U<*unit number*>_C<*channel number*> (n : A to F) to set the compare capture registers. In saw-wave one-shot pulse mode and triangle-wave PWM mode 3, compare capture registers A and B can be set by forcible buffer transfer. The forcible buffer transfer can be executed by R_PG_Timer_Buffer_Force_GPT_U<*unit number*>_C<*channel number*> .
- To start the count operation call R_PG_Timer_StartCount_GPT_U<*unit number*>_C<*channel number*> or R_PG_Timer_SynchronouslyStartCount_GPT_U<*unit number*> after setting the compare capture registers.
- Interrupts of the GPT are set by this function. When the name of the interrupt notification function has been specified in the GUI, if an interrupt occurs in the CPU, the function having the specified name will be called. Create the interrupt notification function as follows:
  void <*name of the interrupt notification function*> (void)
  For the interrupt notification function, note the contents of 5.21, Notes on Notification Functions.

Example

Refer to the example of R_PG_Timer_Set_GPT_U<*unit number*>.

5.10.3 R_PG_Timer_StartCount_GPT_U<*unit number*>_C<*channel number*>

Definition
bool R_PG_Timer_StartCount_GPT_U<*unit number*>_C<*channel number*> (void)
<*unit number*>: 0
<*channel number*>: 0 to 3

Description
Start the GPT count operation

Parameter
None

Return value

true	Setting was made correctly
false	Setting failed

File for output
R_PG_Timer_GPT_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 3

RPDL function
R_GPT_ControlChannel

Details
- Starts the GPT count operation.
- Call R_PG_Timer_Set_GPT_U<*unit number*> and R_PG_Timer_GPT_U<*unit number*>_C<*channel number*> to make the initial settings before calling this function.
- To start the count operation of two or more channels simultaneously, use R_PG_Timer_SynchronouslyStartCount_GPT_U<*unit number*>.

Example
Refer to the example of R_PG_Timer_Set_GPT_U<*unit number*>.

5.10.4 R_PG_Timer_SynchronouslyStartCount_GPT_U<*unit number*>

Definition

bool R_PG_Timer_SynchronouslyStartCount_GPT_U<*unit number*>
(bool gpt0, bool gpt1, bool gpt2, bool gpt3)
<*unit number*>: 0

Description

Start the GPT count operation of two or more channels simultaneously

Parameter

bool gpt0	Count operation of channel 0 (0:Do not start count 1:Start count)
bool gpt1	Count operation of channel 1 (0:Do not start count 1:Start count)
bool gpt2	Count operation of channel 2 (0:Do not start count 1:Start count)
bool gpt3	Count operation of channel 3 (0:Do not start count 1:Start count)

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_Timer_GPT_U<*unit number*>.c
<*unit number*>: 0

RPDL function

R_GPT_ControlUnit

Details

- Starts the GPT count operation of two or more channels simultaneously.
- Call R_PG_Timer_Set_GPT_U<*unit number*> and R_PG_Timer_GPT_U<*unit number*>_C<*channel number*> to make the initial settings before calling this function.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func1(void)
{
      R_PG_Timer_Set_GPT_U0();    // Release GPT form module stop
      R_PG_Timer_Set_GPT_U0_C0();    // Set up GPT0
      R_PG_Timer_Set_GPT_U0_C2();    // Set up GPT2

      R_PG_Timer_SetGTCCR_A_GPT_U0_C0( 0x0000 );    // Set GPT0.GTCCRA
      R_PG_Timer_SetGTCCR_C_GPT_U0_C0( 0x00ff );    // Set GPT0.GTCCRC

      R_PG_Timer_SetGTCCR_A_GPT_U0_C2( 0x0000 );    // Set GPT2.GTCCRA
      R_PG_Timer_SetGTCCR_C_GPT_U0_C2( 0x00ff );    // Set GPT2.GTCCRC
}
void func2(void)
{
      // Start the count operation of GPT0 and 2
      R_PG_Timer_SynchronouslyStartCount_GPT_U0( 1, 0, 1, 0 );
}

void func3(void)
{
      // Halt the count operation of GPT0 and 2
      R_PG_Timer_SynchronouslyHaltCount_GPT_U0( 1, 0, 1, 0 );
}
```

5.10.5 R_PG_Timer_HaltCount_GPT_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_HaltCount_GPT_U<*unit number*>_C<*channel number*> (void)
<*unit number*>: 0
<*channel number*>: 0 to 3

Description

Halt the GPT count operation

Parameter

None

Return value

true	Halting succeeded.
false	Halting failed.

File for output

R_PG_Timer_GPT_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 3

RPDL function

R_GPT_ControlChannel

Details

- Halts the GPT count operation.
- To resume the count operation, call
  R_PG_Timer_StartCount_GPT_U<*unit number*>_C<*channel number*> or
  R_PG_Timer_SynchronouslyStartCount_GPT_U<*unit number*>.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
      // Halt the count operation of GPT0
      R_PG_Timer_HaltCount_GPT_U0_C0();

      // Set counter
      R_PG_Timer_SetCounterValue_GPT_U0_C0( 0xff );

      // Start the count operaion of GPT0
      R_PG_Timer_StartCount_GPT_U0_C0();
}
```

5.10.6 R_PG_Timer_SynchronouslyHaltCount_GPT_U<*unit number*>

Definition

bool R_PG_Timer_SynchronouslyHaltCount_GPT_U<*unit number*>
(bool gpt0, bool gpt1, bool gpt2, bool gpt3)
<*unit number*>: 0

Description

Halt the GPT count operation of two or more channels simultaneously

Parameter

bool gpt0	Count operation of channel 0 (0:Do not stop count 1:Stop count)
bool gpt1	Count operation of channel 1 (0:Do not stop count 1:Stop count)
bool gpt2	Count operation of channel 2 (0:Do not stop count 1:Stop count)
bool gpt3	Count operation of channel 3 (0:Do not stop count 1:Stop count)

Return value

true	Halting succeeded.
false	Halting failed.

File for output

R_PG_Timer_GPT_U<*unit number*>.c
<*unit number*>: 0

RPDL function

R_GPT_ControlUnit

Details

- Halts the GPT count operation of two or more channels simultaneously.
- To resume the count operation, call
  R_PG_Timer_StartCount_GPT_U<*unit number*>_C<*channel number*> or
  R_PG_Timer_SynchronouslyStartCount_GPT_U<*unit number*>.

Example

Refer to the example of R_PG_Timer_SynchronouslyStartCount_GPT_U<*unit number*>.

5.10.7 R_PG_Timer_SetGTCCR_<*GTCCR*>_GPT_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_SetGTCCR_<*GTCCR*>_GPT_U<*unit number*>_C<*channel number*>
(uint16_t gtccr_val)

<*GTCCR*>: A to F
<*unit number*>: 0
<*channel number*>: 0 to 3

Description

Write the value to the compare capture register (GTCCRn n : A to F)

Parameter

uint16_t gtccr_val	The value to be written to the compare capture register

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_Timer_GPT_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 3

RPDL function

R_GPT_ControlChannel

Details

- This function sets the compare capture register (GTCCRn n : A to F) .
- The compare capture registers are not set in
  R_PG_Timer_Set_GPT_U<*unit number*>_C<*channel number*>.
  To write the value to the compare capture registers in the initial setting, use this function.

Example

Refer to the example of R_PG_Timer_Set_GPT_U<*unit number*>.

5.10.8 R_PG_Timer_GetGTCCR_GPT_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_GetGTCCR_GPT_U<*unit number*>_C<*channel number*>
(uint16_t * gtccr_a_val, uint16_t * gtccr_b_val, uint16_t * gtccr_c_val,
uint16_t * gtccr_d_val, uint16_t * gtccr_e_val, uint16_t * gtccr_f_val)

<*unit number*>: 0
<*channel number*>: 0 to 3

Description

Get the value from the compare capture registers (GTCCRA to F)

Parameter

uint16_t * gtccr_a_val	The address of the storage area for compare capture register A value
uint16_t * gtccr_b_val	The address of the storage area for compare capture register B value
uint16_t * gtccr_c_val	The address of the storage area for compare capture register C value
uint16_t * gtccr_d_val	The address of the storage area for compare capture register D value
uint16_t * gtccr_e_val	The address of the storage area for compare capture register E value
uint16_t * gtccr_f_val	The address of the storage area for compare capture register F value

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output

R_PG_Timer_GPT_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 3

RPDL function

R_GPT_ReadChannel

Details

- This function acquires the compare capture register (GTCCRA to F) value.
- Specify the address of storage area for an item to be acquired. Specify 0 for an item that is not required.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

uint16_t  gtccr_a_val, gtccr_c_val;

void func(void)
{
      //Get GTCCRA and GTCCRB value
      R_PG_Timer_GetGTCCR_GPT_U0_C0(
            &gtccr_a_val,     //GTCCRA
            0,                //GTCCRB (not required)
            &gtccr_c_val,     //GTCCRC
            0,                //GTCCRD (not required)
            0,                //GTCCRE (not required)
            0                 //GTCCRF (not required)
      );
}
```

5.10.9 R_PG_Timer_SetCounterValue_GPT_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_SetCounterValue_GPT_U<*unit number*>_C<*channel number*>
(uint16_t counter_val)

<*unit number*>: 0
<*channel number*>: 0 to 3

Description

Set the GPT counter value

Parameter

uint16_t counter_val	The value to be written to the counter

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_Timer_GPT_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 3

RPDL function

R_GPT_ControlChannel

Details

- Set the counter value
- The counter value can be changed only when the counting is stopped.

Example

Refer to the example of R_PG_Timer_HaltCount_GPT_U<*unit number*>_C<*channel number*>.

5.10.10 R_PG_Timer_GetCounterValue_GPT_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_GetCounterValue_GPT_U<*unit number*>_C<*channel number*>
(uint16_t * counter_val)

<*unit number*>: 0
<*channel number*>: 0 to 3

Description

Get the GPT counter value

Parameter

uint16_t * counter_val	The address of the storage area for compare capture register A value

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output

R_PG_Timer_GPT_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 3

RPDL function

R_GPT_ReadChannel

Details

- Get the counter value

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

uint16_t  counter_val;

void func(void)
{
      //Get counter value
      R_PG_Timer_GetCounterValue_GPT_U0_C0( & counter_val );
}
```

5.10.11 R_PG_Timer_SynchronouslyClearCounter_GPT_U<*unit number*>

Definition

bool R_PG_Timer_SynchronouslyClearCounter_GPT_U<*unit number*>
(bool gpt0, bool gpt1, bool gpt2, bool gpt3)
<*unit number*>: 0

Description

Clear the counter of two or more channels simultaneously

Parameter

bool gpt0	GPT0 counter clearing control (0:Do not clear counter 1:Clear counter)
bool gpt1	GPT1 counter clearing control (0:Do not clear counter 1:Clear counter)
bool gpt2	GPT2 counter clearing control (0:Do not clear counter 1:Clear counter)
bool gpt3	GPT3 counter clearing control (0:Do not clear counter 1:Clear counter)

Return value

true	Clearing succeeded
false	Clearing failed

File for output

R_PG_Timer_GPT_U<*unit number*>.c
<*unit number*>: 0

RPDL function

R_GPT_ControlUnit

Details

- Clear the GPT counter of two or more channels simultaneously.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

uint16_t  counter_val;

void func(void)
{
      //Clear the counter of GPT0 and GPT2
      R_PG_Timer_SynchronouslyClearCounter_GPT_U0( 1, 0, 1, 0 );
}
```

5.10.12 R_PG_Timer_SetCycle_GPT_U<*unit number*>_C<*channel number*>

Definition bool R_PG_Timer_SetCycle_GPT_U<*unit number*>_C<*channel number*> (uint16_t gtpr_val)

<*unit number*>: 0
<*channel number*>: 0 to 3

Description Set the timer cycle setting register (GTPR)

Parameter

uint16_t gtpr_val	The value to be written to the timer cycle setting register

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_Timer_GPT_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 3

RPDL function R_GPT_ControlChannel

Details
- This function sets the timer cycle setting register (GTPR).

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
      //Set timer cycle register
      R_PG_Timer_SetCycle_GPT_U0_C0( 0x6000 );
}
```

5.10.13 R_PG_Timer_SetBuffer_Cycle_GPT_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_SetBuffer_Cycle_GPT_U<*unit number*>_C<*channel number*>
(uint16_t gtpbr_val)

<*unit number*>: 0
<*channel number*>: 0 to 3

Description

Set the timer cycle setting buffer register (GTPBR)

Conditions for output

The buffer operation of the cycle setting register (GTPR) is selected

Parameter

uint16_t gtpbr_val	The value to be written to the timer cycle setting buffer register

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_Timer_GPT_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 3

RPDL function

R_GPT_ControlChannel

Details

- This function sets the timer cycle setting buffer register (GTPBR).

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
      //Set timer cycle setting buffer register
      R_PG_Timer_SetBuffer_Cycle_GPT_U0_C0( 0x5000 );
}
```

5.10.14 R_PG_Timer_SetDoubleBuffer_Cycle_GPT_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_SetDoubleBuffer_Cycle_GPT_U<*unit number*>_C<*channel number*>
(uint16_t gtpdbr_val)

<*unit number*>: 0
<*channel number*>: 0 to 3

Description

Set the timer cycle setting double-buffer register (GTPDBR)

Conditions for output

The double buffer operation of the cycle setting register (GTPR) is selected

Parameter

uint16_t gtpdbr_val	The value to be written to the timer cycle setting double-buffer register

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_Timer_GPT_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 3

RPDL function

R_GPT_ControlChannel

Details

- This function sets the timer cycle setting double-buffer register (GTPDBR).

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
      //Set timer cycle setting double-buffer register
      R_PG_Timer_SetDoubleBuffer_Cycle_GPT_U0_C0( 0x4000 );
}
```

5.10.15 R_PG_Timer_SetAD_GPT_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_SetAD_GPT_U<*unit number*>_C<*channel number*>
(uint16_t gtadtra_val, uint16_t gtadtrb_val)

<*unit number*>: 0
<*channel number*>: 0 to 3

Description

Set the A/D converter start request timing register A, B (GTADTRA, B)

Conditions for output

The A/D converter start request is enabled

Parameter

uint16_t gtadtra_val	The value to be written to GTADTRA
uint16_t gtadtrb_val	The value to be written to GTADTRB

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_Timer_GPT_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 3

RPDL function

R_GPT_ControlChannel

Details

- This function sets the A/D converter start request timing register A, B (GTADTRA, B).

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
    // Set the A/D converter start request timing registers
    R_PG_Timer_SetAD_GPT_U0_C0(
        0x3000,    // A/D converter start request timing register A (GTADTRA)
        0x2000     // A/D converter start request timing register B (GTADTRB)
    );
}
```

5.10.16 R_PG_Timer_SetBuffer_AD_GPT_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_SetBuffer_AD_GPT_U<*unit number*>_C<*channel number*>
(uint16_t gtadtbra_val, uint16_t gtadtbrb_val)

<*unit number*>: 0
<*channel number*>: 0 to 3

Description

Set the A/D converter start request timing buffer register A, B (GTADTBRA, GTADTBRB)

Conditions for output

The buffer transfer of A/D converter start request timing register is enabled

Parameter

uint16_t gtadtbra_val	The value to be written to GTADTBRA
uint16_t gtadtbrb_val	The value to be written to GTADTBRB

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_Timer_GPT_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 3

RPDL function

R_GPT_ControlChannel

Details

- This function sets the A/D converter start request timing buffer register A, B (GTADTBRA, GTADTBRB).

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
    // Set the A/D converter start request timing buffer registers
    R_PG_Timer_SetBuffer_AD_GPT_U0_C0(
        0x6000,    // A/D converter start request timing buffer register A (GTADTBRA)
        0x3000     // A/D converter start request timing buffer register B (GTADTBRB)
    );
}
```

5.10.17 R_PG_Timer_SetDoubleBuffer_AD_GPT_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_SetDoubleBuffer_AD_GPT_U<*unit number*>_C<*channel number*>
(uint16_t gtadtdbra_val, uint16_t gtadtdbrb_val)

<*unit number*>: 0
<*channel number*>: 0 to 3

Description

Set the A/D converter start request timing double-buffer register A, B (GTADTDBRA, GTADTDBRB)

Conditions for output

The double-buffer transfer of A/D converter start request timing register is enabled

Parameter

uint16_t gtadtdbra_val	The value to be written to GTADTDBRA
uint16_t gtadtdbrb_val	The value to be written to GTADTDBRB

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_Timer_GPT_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 3

RPDL function

R_GPT_ControlChannel

Details

- This function sets the A/D converter start request timing double-buffer register A, B (GTADTDBRA, GTADTDBRB).

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
      // Set the A/D converter start request timing double-buffer register
      R_PG_Timer_SetDoubleBuffer_AD_GPT_U0_C0(
            0x8000,    // GTADTDBRA
            0x4000     // GTADTDBRB
      );
}
```

5.10.18 R_PG_Timer_SetBuffer_GTDV<*U/D*>_GPT_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_SetBuffer_GTDVU_GPT_U<*unit number*>_C<*channel number*>
(uint16_t gtdbu_val)

bool R_PG_Timer_SetBuffer_GTDVD_GPT_U<*unit number*>_C<*channel number*>
(uint16_t gtdbd_val)

<*unit number*>: 0
<*channel number*>: 0 to 3

Description

Set the timer dead time buffer register U, D (GTDBU, GTDBD)

Conditions for output

Automatic addition of dead time is enabled

Parameter

uint16_t gtdbu_val	The value to be written to GTDBU
uint16_t gtdbd_val	The value to be written to GTDBD

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_Timer_GPT_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 3

RPDL function

R_GPT_ControlChannel

Details

- This function sets the timer dead time buffer register U, D (GTDBU, GTDBD) that are the buffer registers of the timer dead time value register U, D (GTDVU, GTDVD),.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
    // Set the timer dead time buffer register U
    R_PG_Timer_SetBuffer_GTDVU_GPT_U0_C0( 0x500 );

    // Set the timer dead time buffer register D
    R_PG_Timer_SetBuffer_GTDVD_GPT_U0_C0( 0x300 );
}
```

5.10.19 R_PG_Timer_GetRequestFlag_GPT_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_GetRequestFlag_GPT_U<*unit number*>_C<*channel number*>
(bool * cm_ic_a, bool * cm_ic_b, bool * cm_c, bool * cm_d,
bool * cm_e, bool * cm_f, bool * ov, bool * un, bool * dt_error)

<*unit number*>: 0
<*channel number*>: 0 to 3

Description

Get and clear the GPT interrupt flag

Parameter

bool * cm_ic_a	The address of the storage area for the compare match/input capture A flag
bool * cm_ic_b	The address of the storage area for the compare match/input capture B flag
bool * cm_c	The address of the storage area for the compare match/input capture C flag
bool * cm_d	The address of the storage area for the compare match/input capture D flag
bool * cm_e	The address of the storage area for the compare match/input capture E flag
bool * cm_f	The address of the storage area for the compare match/input capture F flag
bool * ov	The address of the storage area for the overflow flag
bool * un	The address of the storage area for the underflow flag
bool * dt_error	The address of the storage area for the dead time error flag

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output

R_PG_Timer_GPT_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 3

RPDL function

R_GPT_ReadChannel

Details

- This function acquires the interrupt flags of GPT.
- All flags will be cleared in this function.
- Specify the address of storage area for the flags to be acquired. Specify 0 for a flag that is not required.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

bool cm_ic_a, ov;

void func(void)
{
    //Get compare match/input capture A flag and overflow flag
    R_PG_Timer_GetRequestFlag_GPT_U0_C0(
        &cm_ic_a,   // Compare match/input capture A flag
        0,          // Compare match/input capture B flag (not required )
        0,          // Compare match/input capture C flag (not required )
        0,          // Compare match/input capture D flag (not required )
        0,          // Compare match/input capture E flag (not required )
        0,          // Compare match/input capture F flag (not required )
        &ov,        // Overflow flag
        0,          // Underflow flag (not required )
        0           // dead time error flag (not required )
    }
}
```

5.10.20 R_PG_Timer_GetRequestFlag_GPT_U<*unit number*>

Definition

bool R_PG_Timer_GetRequestFlag_GPT_U<*unit number*>
(bool * loco_rising, bool * loco_deviation, bool * loco_ov,
bool * ext_rising, bool * ext_falling)

<*unit number*>: 0
<*channel number*>: 0 to 3

Description

Get and clear the GPT interrupt flags of LOCO count function and external trigger

Parameter

bool * loco_rising	The address of the storage area for the frequency-divided LOCO clock rise interrupt request flag
bool * loco_deviation	The address of the storage area for the fLOCO count value deviation exceedance interrupt request flag
bool * loco_ov	The address of the storage area for the LCNT overflow interrupt request flag
bool * ext_rising	The address of the storage area for the external trigger rising input interrupt request flag
bool * ext_falling	The address of the storage area for the external trigger falling input interrupt request flag

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output

R_PG_Timer_GPT_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 3

RPDL function

R_GPT_ReadUnit

Details

- This function acquires and clears the interrupt flags of LOCO count function and external
- trigge.
- All flags will be cleared in this function.
  Specify the address of storage area for the flags to be acquired. Specify 0 for a flag that is not required.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

bool loco_deviation, ext_rising;

void func(void)
{
    //Get fLOCO count deviation exceedance interrupt flag and
    //external trigger rising input interrupt flag

    R_PG_Timer_GetRequestFlag_GPT_U0 (
        0,                  // frequency-divided LOCO clock rise interrupt flag (not required)
        &loco_deviation,    // fLOCO count deviation exceedance interrupt flag
        0,                  // LCNT overflow interrupt flag (not required)
        &ext_rising,        // External trigger rising input interrupt flag
        0                   // External trigger falling input interrupt flag (not required)
    );
}
```

5.10.21 R_PG_Timer_GetCounterStatus_GPT_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_GetCounterStatus_GPT_U<*unit number*>_C<*channel number*>
(bool * active, bool * up)

<*unit number*>: 0
<*channel number*>: 0 to 3

Description

Get the counter status

Parameter

bool * active	The address of the storage area for the count start bit (0 : Count operation is performed 1 : Count operation is stopped)
bool * up	The address of the storage area for the count direction flag (0 : Downward 1 : Upward)

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output

R_PG_Timer_GPT_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 3

RPDL function

R_GPT_ReadChannel

Details

- This function acquires the count start bit and count direction flag.
- Specify the address of storage area for the flags to be acquired. Specify 0 for a flag that is not required.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

bool up;

void func(void)
{
    //Get count direction
    R_PG_Timer_GetCounterStatus_GPT_U0_C0 (
        0,                  // Count start bit (not required)
        & up                // Count direction flag
    );
}
```

5.10.22 R_PG_Timer_BufferEnable_GPT_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_BufferEnable_GPT_U<*unit number*>_C<*channel number*>
(bool gtccr, bool gtpr, bool gtadtr, bool gtdv)

<*unit number*>: 0
<*channel number*>: 0 to 3

Description

Enable the buffer operation

Parameter

bool gtccr	Buffer operation setting of compare capture register GTCCRA, GTCCRC, GTCCRD and GTCCRB, GTCCRE, GTCCRF. (0:Do not enable buffer operation 1:Enable buffer operation)
bool gtpr	Buffer operation setting of timer cycle setting register (GTPR) and timer cycle setting buffer register (GTPBR) (0:Do not enable buffer operation 1:Enable buffer operation)
bool gtadtr	Buffer operation setting of A/D converter start request timing register (GTADTRA), A/D converter start request timing buffer register (GTADTBRA) and A/D converter start request timing double-buffer register (GTADTDBRA) (0:Do not enable buffer operation 1:Enable buffer operation)
bool gtdv	Buffer operation setting of timer dead time value register U, D (GTDVU, GTDVD) and timer dead time value register U,D (GTDBU, GTDBD) (0:Do not enable buffer operation 1:Enable buffer operation)

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_Timer_GPT_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 3

RPDL function

R_GPT_ControlChannel

Details

- This function enables the buffer operation.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
      // Enable buffer operation of GTCCRA, C, D and GTCCRB, E, F.
      R_PG_Timer_BufferEnable_GPT_U0_C0 (
         1,   //Enable buffer operation of GTCCRA, C, D and GTCCRB, E, F
         0,   //Do not enable buffer operation of GTPR
         0,   //Do not enable buffer operation of GTADTRA
         0    //Do not enable buffer operation of GTDVU and GTDVD
      );
}
```

5.10.23 R_PG_Timer_BufferDisable_GPT_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_BufferDisable_GPT_U<*unit number*>_C<*channel number*>
(bool gtccr, bool gtpr, bool gtadtr, bool gtdv)

<*unit number*>: 0
<*channel number*>: 0 to 3

Description

Disable the buffer operation

Parameter

bool gtccr	Buffer operation setting of compare capture register GTCCRA, GTCCRC, GTCCRD and GTCCRB, GTCCRE, GTCCRF. (0:Do not disable buffer operation 1:Disable buffer operation)
bool gtpr	Buffer operation setting of timer cycle setting register (GTPR) and timer cycle setting buffer register (GTPBR) (0:Do not disable buffer operation 1:Disable buffer operation)
bool gtadtr	Buffer operation setting of A/D converter start request timing register (GTADTRA), A/D converter start request timing buffer register (GTADTBRA) and A/D converter start request timing double-buffer register (GTADTDBRA) (0:Do not disable buffer operation 1:Disable buffer operation)
bool gtdv	Buffer operation setting of timer dead time value register U, D (GTDVU, GTDVD) and timer dead time value register U,D (GTDBU, GTDBD) (0:Do not disable buffer operation 1:Disable buffer operation)

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_Timer_GPT_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 3

RPDL function

R_GPT_ControlChannel

Details

- This function disables the buffer operation.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
      // Disable buffer operation of GTCCRA, C, D and GTCCRB, E, F.
      R_PG_Timer_BufferDisable_GPT_U0_C0 (
         1,   //Disable buffer operation of GTCCRA, C, D and GTCCRB, E, F
         0,   //Do not disable buffer operation of GTPR
         0,   //Do not disable buffer operation of GTADTRA
         0    //Do not disable buffer operation of GTDVU and GTDVD
      );
}
```

5.10.24 R_PG_Timer_Buffer_Force_GPT_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_Buffer_Force_GPT_U<*unit number*>_C<*channel number*> (void)

<*unit number*>: 0
<*channel number*>: 0 to 3

Description

Execute forcible buffer transfer

Parameter

None

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_Timer_GPT_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 3

RPDL function

R_GPT_ControlChannel

Details

- Execute forcible buffer transfer of GTCCRA and GTCCRB.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
      R_PG_Timer_Set_GPT_U0();                    // Release GPT form module stop
      R_PG_Timer_Set_GPT_U0_C0();                 // Set up GPT0
      R_PG_Timer_SetGTCCR_C_GPT_U0_C0( 0x6000 );  // Set GTCCRC
      R_PG_Timer_SetGTCCR_D_GPT_U0_C0( 0x3000);   // Set GTCCRD
      R_PG_Timer_SetGTCCR_E_GPT_U0_C0( 0x8000 );  // Set GTCCRE
      R_PG_Timer_SetGTCCR_F_GPT_U0_C0( 0x4000 );  // Set GTCCRF
      R_PG_Timer_Buffer_Force_GPT_U0_C0();        //Execute forcible transfer
      R_PG_Timer_StartCount_GPT_U0_C0();          // Start the count operaion
}
```

5.10.25 R_PG_Timer_CountDirection_Down_GPT_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_CountDirection_Down_GPT_U<*unit number*>_C<*channel number*>
(bool force)

<*unit number*>: 0
<*channel number*>: 0 to 3

Description

Set the count direction to down-counting

Parameter

bool force	Forcible count direction setting (0:Do not set forcibly 1:Set forcibly)

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_Timer_GPT_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 3

RPDL function

R_GPT_ControlChannel

Details

- This function sets the count direction to down-counting.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
      // Set count direction to down ( Do not set forcibly)
      R_PG_Timer_CountDirection_Down_GPT_U0_C0( 0 );
}
```

5.10.26 R_PG_Timer_CountDirection_Up_GPT_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_CountDirection_Up_GPT_U<*unit number*>_C<*channel number*>
(bool force)

<*unit number*>: 0
<*channel number*>: 0 to 3

Description

Set the count direction to up-counting

Parameter

bool force	Forcible count direction setting (0:Do not set forcibly 1:Set forcibly)

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_Timer_GPT_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 3

RPDL function

R_GPT_ControlChannel

Details

- This function sets the count direction to up-counting.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
      // Set count direction to up ( Set forcibly )
      R_PG_Timer_CountDirection_Up_GPT_U0_C0( 1 );
}
```

5.10.27 R_PG_Timer_SoftwareNegate_GPT_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_SoftwareNegate_GPT_U<*unit number*>_C<*channel number*>
(bool on)

<*unit number*>: 0
<*channel number*>: 0 to 3

Description

Control GTIOCnA and GTIOCnB pin output negation by software (n:Channel number)

Conditions for output

GTIOCnA or GTIOCnB pin output negation control is enabled and software control is selected for the negation source

Parameter

bool on	Output value of the negation source (1:ON 0:OFF)

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_Timer_GPT_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0
<*channel number*>: 0 to 3

RPDL function

R_GPT_ControlChannel

Details

- This function controls the negation of GTIOCnA and GTIOCnB pin output. (n:Channel number)

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
      // Set the value of the negation source to 1
      R_PG_Timer_CountDirection_Up_GPT_U0_C0( 1 );
}
```

5.10.28 R_PG_Timer_StartCount_LOCO_GPT_U<*unit number*>

Definition: bool R_PG_Timer_StartCount_LOCO_GPT_U<*unit number*> (void)
<*unit number*>: 0

Description: Start the LOCO count

Conditions for output: The LOCO count function is enabled

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Timer_GPT_U<*unit number*>.c
<*unit number*>: 0

RPDL function: R_GPT_ControlUnit

Details:

- Starts the LOCO count

Example:

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
    // Set up IWDT and start the count operation
    R_PG_Timer_Set_IWDT();
    R_PG_Timer_RefreshCounter_IWDT();

    // Release GPT form module stop and set up LOCO count function
    R_PG_Timer_Set_GPT_U0();

    // Start the LOCO count
    R_PG_Timer_StartCount_LOCO_GPT_U0();
}
```

5.10.29 R_PG_Timer_HaltCount_LOCO_GPT_U<*unit number*>

Definition: bool R_PG_Timer_HaltCount_LOCO_GPT_U<*unit number*> (void)
<*unit number*>: 0

Description: Halt the LOCO count

Conditions for output: The LOCO count function is enabled

Parameter: None

Return value:

true	Halting succeeded.
false	Halting failed.

File for output: R_PG_Timer_GPT_U<*unit number*>.c
<*unit number*>: 0

RPDL function: R_GPT_ControlUnit

Details:

- Halts the LOCO count

Example:

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
    // Halt the LOCO count
    R_PG_Timer_HaltCount_LOCO_GPT_U0();
}
```

5.10.30 R_PG_Timer_ClearCounter_LOCO_GPT_U<*unit number*>

Definition: bool R_PG_Timer_ClearCounter_LOCO_GPT_U<*unit number*> (void)
<*unit number*>: 0

Description: Clear the LOCO count value register

Conditions for output: The LOCO count function is enabled

Parameter: None

Return value:

true	Clearing succeeded
false	Clearing failed

File for output: R_PG_Timer_GPT_U<*unit number*>.c
<*unit number*>: 0

RPDL function: R_GPT_ControlUnit

Details:

- Clears the LOCO count value register

Example:

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
    // Clear the LOCO count value register
    R_PG_Timer_ClearCounter_LOCO_GPT_U0();
}
```

5.10.31 R_PG_Timer_InitialiseCountResultValue_LOCO_GPT_U<*unit number*>

Definition
bool R_PG_Timer_InitialiseCountResultValue_LOCO_GPT_U<*unit number*> (void)
<*unit number*>: 0

Description
Initialize the LOCO count result registers

Conditions for output
The LOCO count function is enabled

Parameter
None

Return value

true	Setting was made correctly
false	Setting failed

File for output
R_PG_Timer_GPT_U<*unit number*>.c
<*unit number*>: 0

RPDL function
R_GPT_ControlUnit

Details
- Initialise the LOCO count result registers LCNT01 to LCNT15 using the LCNT00 value.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
      // Initialize the LOCO count result registers
      R_PG_Timer_InitialiseCountResultValue_LOCO_GPT_U0();
}
```

5.10.32 R_PG_Timer_GetCounterValue_LOCO_GPT_U<*unit number*>

Definition

bool R_PG_Timer_GetCounterValue_LOCO_GPT_U<*unit number*>
(uint16_t * loco_counter_val)
<*unit number*>: 0

Description

Get the value of the LOCO count value register

Conditions for output

The LOCO count function is enabled

Parameter

uint16_t * loco_counter_val	The address of the storage area for the LOCO count value register

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output

R_PG_Timer_GPT_U<*unit number*>.c
<*unit number*>: 0

RPDL function

R_GPT_ReadUnit

Details

- Gets the value of the LOCO count value register.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

uint16_t loco_counter_val;

void func(void)
{
    // Get the value of the LOCO count value register
    R_PG_Timer_GetCounterValue_LOCO_GPT_U0( &loco_counter_val );
}
```

5.10.33 R_PG_Timer_GetCounterAverageValue_LOCO_GPT_U<*unit number*>

Definition

bool R_PG_Timer_GetCounterAverageValue_LOCO_GPT_U<*unit number*>
(uint16_t * loco_counter_ave_val)
<*unit number*>: 0

Description

Get the LOCO count result average value

Conditions for output

The LOCO count function is enabled

Parameter

uint16_t * loco_counter_ave_val	The address of the storage area for the LOCO count result average value

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output

R_PG_Timer_GPT_U<*unit number*>.c
<*unit number*>: 0

RPDL function

R_GPT_ReadUnit

Details

- Get the LOCO count result average register value.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

uint16_t    loco_counter_ave_val;

void func(void)
{
      // Get the LOCO count result average value
      R_PG_Timer_GetCounterAverageValue_LOCO_GPT_U0( & loco_counter_ave_val );
}
```

5.10.34 R_PG_Timer_GetCountResultValue_LOCO_GPT_U<*unit number*>

Definition

bool R_PG_Timer_GetCountResultValue_LOCO_GPT_U<*unit number*>
(uint16_t * loco_count_result_val)
<*unit number*>: 0

Description

Get the LOCO count result registers value

Conditions for output

The LOCO count function is enabled

Parameter

uint16_t * loco_count_result_val	A pointer to where the LOCO count result registers value shall be stored. (Provide space for 32-byte values)

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output

R_PG_Timer_GPT_U<*unit number*>.c
<*unit number*>: 0

RPDL function

R_GPT_ReadUnit

Details

- Get the LOCO count result registers (LCNT00～LCNT15) value.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

uint16_t    loco_count_result_val[16];

void func(void)
{
      // Get the value of LOCO count result registers
      R_PG_Timer_GetCountResultValue_LOCO_GPT_U0( loco_count_result_val );
}
```

5.10.35 R_PG_Timer_SetPermissibleDeviation_LOCO_GPT_U<*unit number*>

Definition

bool R_PG_Timer_SetPermissibleDeviation_LOCO_GPT_U<*unit number*>
(uint16_t maximum_val, uint16_t minimum_val)
<*unit number*>: 0

Description

Set the LOCO count upper/lower permissible deviation value

Conditions for output

The LOCO count function is enabled and the LOCO count value deviation exceedance interrupt is enabled.

Parameter

uint16_t maximum_val	The value to be written to the LOCO count upper permissible deviation register
uint16_t minimum_val	The value to be written to the LOCO count lower permissible deviation register

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_Timer_GPT_U<*unit number*>.c
<*unit number*>: 0

RPDL function

R_GPT_ControlUnit

Details

- Sets the LOCO count upper/lower permissible deviation value.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
    // Set the LOCO count upper/lower permissible deviation value
    R_PG_Timer_SetPermissibleDeviation_LOCO_GPT_U0(
        0x10    // Upper permissible deviation
        0x10    // Lower permissible deviation
    );
}
```

5.10.36 R_PG_Timer_AdjustEdgeDelay_GPT_ U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_AdjustEdgeDelay_GPT_U<*unit number*>_C<*channel number*>
(uint8_t GTIOCA_Rising_Delay, uint8_t GTIOCA_Falling_Delay,
uint8_t GTIOCB_Rising_Delay, uint8_t GTIOCB_Falling_Delay)
<*unit number*>: 0
<channel number>:0 to 3

Description

Update the delay times

Parameter

uint8_t GTIOCA_Rising_Delay	The value to be written to the GTIOCA Rising Output Delay Register (GTDLYRA)(1-31:Delay setting, 0:No delay)
uint8_t GTIOCA_Falling_Delay	The value to be written to the GTIOCA Falling Output Delay Register (GTDLYFA) (1-31:Delay setting, 0:No delay)
uint8_t GTIOCB_Rising_Delay	The value to be written to the GTIOCB Rising Output Delay Register (GTDLYRB) (1-31:Delay setting, 0:No delay)
uint8_t GTIOCB_Falling_Delay	The value to be written to the GTIOCB Falling Output Delay Register (GTDLYFB) (1-31:Delay setting, 0:No delay)

Return value

true	Setting was made correctly
False	Setting failed

File for output

R_PG_Timer_GPT_U<*unit number*>_<*channel number*>.c
<*unit number*>: 0
<*channel number*>:0 to 3

RPDL function

R_GPT_EdgeDelay_Control

Details

- Update the delay times.
  Call R_PG_Timer_EnableEdgeDelay_GPT_U<*unit number*> to enable the delay times settings,

Example

```
//Include “R_PG_<PDG project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
      // Update the delay times
      R_PG_Timer_AdjustEdgeDelay_GPT_U0_C0(5, 10, 5, 10);
      // Enable delay times setteings.
      R_PG_Timer_EnableEdgeDelay_GPT_U0(1, 0, 0, 0);
}
```

5.10.37 R_PG_Timer_EnableEdgeDelay_GPT_ U<*unit number*>

Definition

bool R_PG_Timer_EnableEdgeDelay_GPT_U<*unit number*>
(bool C0_Enable, bool C1_Enable, bool C2_Enable, bool C3_Enable)
<*unit number*>: 0

Description

Enable the delay circuit

Parameter

bool C0_Enable	Delay generation control for GPT0 (1:Enable 0:No change)
bool C1_Enable	Delay generation control for GPT1 (1:Enable 0:No change)
bool C2_Enable	Delay generation control for GPT2 (1:Enable 0:No change)
bool C4_Enable	Delay generation control for GPT3 (1:Enable 0:No change)

Return value

true	Setting was made correctly
False	Setting failed

File for output

R_PG_Timer_GPT_U<*unit number*>.c
<*unit number*>: 0

RPDL function

R_GPT_EdgeDelay_Create

Details

- Enable the delay circuit.
  Call R_PG_Timer_DisableEdgeDelay_GPT_U<*unit number*> to Disable the delay times settings,

Example

```
//Include “R_PG_<PDG project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
    // Update the delay times
    R_PG_Timer_AdjustEdgeDelay_GPT_U0_C0(5, 10, 5, 10);
    // Enable delay times setteings.
    R_PG_Timer_EnableEdgeDelay_GPT_U0(1, 0, 0, 0);
}
```

5.10.38 R_PG_Timer_DisableEdgeDelay_GPT_ U<*unit number*>

Definition

bool R_PG_Timer_DisableEdgeDelay_GPT_U<*unit number*>
(bool C0_Disable, bool C1_Disable, bool C2_Disable, bool C3_Disable)
<*unit number*>: 0

Description

Disable the delay circuit

Parameter

bool C0_Disable	Delay generation control for GPT0 (1:Disable 0:No change)
bool C1_Disable	Delay generation control for GPT1 (1:Disable 0:No change)
bool C2_Disable	Delay generation control for GPT2 (1:Disable 0:No change)
bool C4_Disable	Delay generation control for GPT3 (1:Disable 0:No change)

Return value

true	Setting was made correctly
False	Setting failed

File for output

R_PG_Timer_GPT_U<*unit number*>.c
<*unit number*>: 0

RPDL function

R_GPT_EdgeDelay_Create

Details

- Disable the delay circuit.
  Call R_PG_Timer_EnableEdgeDelay_GPT_U<*unit number*> to Enable the delay times settings,

Example

```
//Include “R_PG_<PDG project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
    // Disable delay times setteings.
    R_PG_Timer_DisableEdgeDelay_GPT_U0(1, 0, 0, 0);
}
```

5.10.39 R_PG_Timer_StopModule_GPT_U<*unit number*>

Definition

bool R_PG_Timer_StopModule_GPT_U<*unit number*> (void)

<*unit number*>: 0

Description

Shut down the GPT

Parameter

None

Return value

true	Shutting down succeeded
False	Shutting down failed

File for output

R_PG_Timer_GPT_U<*unit number*>.c

<*unit number*>: 0

RPDL function

R_GPT_Destroy

Details

- Stops a GPT and places it in the module-stop state. If two or more channels are running when this function is called, all channels will be stopped. Call R_PG_Timer_HaltCount_GPT_U<*unit number*>_C<*channel number*> or R_PG_Timer_SynchronouslyHaltCount_GPT_U<*unit number*> to stop a single channel.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
      // Shut down the GPT
      R_PG_Timer_StopModule_GPT_U0();
}
```

5.11 Compare Match Timer (CMT)

5.11.1 R_PG_Timer_Start_CMT_U<*unit number*>_C<*channel number*>

Definition: bool R_PG_Timer_Start_CMT_U<*unit number*>_C<*channel number*> (void)
<*unit number*>: 0 or 1
<*channel number*>: 0 to 3

Description: Set up the CMT and start the count operation

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Timer_CMT_U<*unit number*>.c
<*unit number*>: 0 and 1

RPDL function: R_CMT_Create

Details:

- Releases the CMT from the module-stop, makes initial settings, and starts the CMT counting.
- Interrupts of the CMT are set by this function. When the name of the interrupt notification function has been specified in the GUI, if an interrupt occurs in the CPU, the function having the specified name will be called. Create the interrupt notification function as follows:
  void <*name of the interrupt notification function*> (void)
  For the interrupt notification function, note the contents of 5.21, Notes on Notification Functions.

Example: A case where the setting is made as follows.

- Cmt0IntFunc was specified as a compare match interrupt notification function name

```
#include "R_PG_default.h" //Include "R_PG_<project name>.h" to use this function.

void func(void)
{
      R_PG_Timer_Start_CMT_U0_C0 (); //Set up the CMT0 and start the count operation
}

void Cmt0IntFunc (void)
{
      func_cmt0();       //Processing in response to a compare match interrupt
}
```

5.11.2 R_PG_Timer_HaltCount_CMT<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_HaltCount_CMT_U<*unit number*>_C<*channel number*> (void)

<*unit number*>: 0 or 1

<*channel number*>: 0 to 3

Description

Halt the CMT count operation

Parameter

None

Return value

true	Halting succeeded.
false	Halting failed.

File for output

R_PG_Timer_CMT_U<*unit number*>.c

<*unit number*>: 0 or 1

RPDL function

R_CMT_Control

Details

- Halts the CMT count operation.
- To resume the count operation, call the following function.
  R_PG_Timer_ResumeCount_CMT_U<*unit number*>_C<*channel number*>

Example

A case where the setting is made as follows.

- CMT unit 0 channel 0 was set up
- Cmt0IntFunc was specified as the compare match interrupt notification function name

```
#include "R_PG_default.h" //Include "R_PG_<project name>.h" to use this function.

void func(void)
{
    R_PG_Timer_Start_CMT_U0_C0();    //Set up the CMT0 and start the count operation
}

void Cmt0IntFunc(void)
{
    //Halt the CMT0 count operation
    R_PG_Timer_HaltCount_CMT_U0_C0();

    func_cmt0();    //Processing in response to a compare match interrupt

    //Resume the CMT0 count operation
    R_PG_Timer_ResumeCount_CMT_U0_C0();
}
```

5.11.3 R_PG_Timer_ResumeCount_CMT_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_ResumeCount_CMT_U<*unit number*>_C<*channel number*> (void)

<*unit number*>: 0 or 1

<*channel number*>: 0 to 3

Description

Resume the CMT count operation

Parameter

None

Return value

true	Resuming count succeeded.
false	Resuming count failed.

File for output

R_PG_Timer_CMT_U<*unit number*>.c

<*unit number*>: 0 or 1

RPDL function

R_CMT_Control

Details

- Resumes the CMT count operation that was halted by R_PG_Timer_HaltCount_CMT_U<*unit number*>_C<*channel number*>.

Example

A case where the setting is made as follows.

- CMT unit 0 channel 0 was set up
- Cmt0IntFunc was specified as the compare match interrupt notification function name

```
#include "R_PG_default.h" //Include "R_PG_<project name>.h" to use this function.

void func(void)
{
    R_PG_Timer_Start_CMT_U0_C0();    //Set up the CMT0 and start the count operation
}

void Cmt0IntFunc(void)
{
    //Halt the CMT0 count operation
    R_PG_Timer_HaltCount_CMT_U0_C0();

    func_cmt0();    //Processing in response to a compare match interrupt

    //Resume the CMT0 count operation
    R_PG_Timer_ResumeCount_CMT_U0_C0();
}
```

5.11.4 R_PG_Timer_GetCounterValue_CMT_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_GetCounterValue_CMT_U<*unit number*>_C<*channel number*>
(uint16_t * counter_val)
<*unit number*>: 0 or 1
<*channel number*>: 0 to 3

Description

Acquire the CMT counter value

Parameter

uint16_t * counter_val	Destination for the storage of the counter value

Return value

true	Acquisition of the counter value succeeded.
false	Acquisition of the counter value failed.

File for output

R_PG_Timer_CMT_U<*unit number*>.c
<*unit number*>: 0 or 1

RPDL function

R_CMT_Read

Details

- Acquires the counter value of a CMT.

Example

A case where the setting is made as follows.

- CMT unit 0 channel 0 was set up

```
#include "R_PG_default.h" //Include "R_PG_<project name>.h" to use this function.
uint16_t counter_val
void func1(void)
{
    R_PG_Timer_Start_CMT_U0_C0();    //Set up the CMT0 and start the count operation
}
unt16_t func2(void)
{
    uint16_t data;
    // Acquire the value of a CMT0 counter
    R_PG_Timer_GetCounterValue_CMT_U0_C0( &counter_val );
    return data;
}
```

5.11.5 R_PG_Timer_SetCounterValue_CMT_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_SetCounterValue_CMT_U<*unit number*>_C<*channel number*>
(uint16_t counter_val)
<*unit number*>: 0 or 1
<*channel number*>: 0 to 3

Description

Set the CMT counter value

Parameter

uint16_t counter_val	Value to be written to the counter

Return value

true	Setting of the counter value succeeded.
false	Setting of the counter value failed.

File for output

R_PG_Timer_CMT_U<*unit number*>.c
<*unit number*>: 0 or 1

RPDL function

R_CMT_Control

Details

- Set the counter value of a CMT.

Example

A case where the setting is made as follows.

- CMT unit 0 channel 0 was set up

```
#include "R_PG_default.h" //Include "R_PG_<project name>.h" to use this function.
void func1(void)
{
    R_PG_Timer_Start_CMT_U0_C0();    //Set up the CMT0 and start the count operation
}
void func2(void)
{
    R_PG_Timer_SetCounterValue_CMT_U0_C0( 0 ); // Set the value of a CMT0 counter
    return;
}
```

5.11.6 R_PG_Timer_StopModule_CMT_U<*unit number*>

Definition: bool R_PG_Timer_StopModule_CMT_U<*unit number*> (void)
<*unit number*>: 0 or 1

Description: Shut down the CMT unit

Parameter: None

Return value:

true	Shutting down succeeded
false	Shutting down failed

File for output: R_PG_Timer_CMT_U<*unit number*>.c
<*unit number*>: 0 or 1

RPDL function: R_CMT_Destroy

Details:

- Stops a CMT unit and places it in the module-stop state per unit. If both CMT0 and CMT1 of unit 0 (or both CMT2 and CMT3 of unit 1) are running when this function is called, both channels are stopped. Call the following function to stop a single channel.
  R_PG_Timer_HaltCount_CMT_U<*unit number*>_C<*channel number*>

Example: A case where the setting is made as follows.

- CMT unit 0 channel 0 was set up
  Cmt0IntFunc was specified as the compare match interrupt notification function name

```
#include "R_PG_default.h" //Include "R_PG_<project name>.h" to use this function.

void func(void)
{
    R_PG_Timer_Start_CMT_U0_C0();    //Set up the CMT0 and start the count operation
}

void Cmt0IntFunc(void)
{
    func_cmt();        //Processing in response to a compare match interrupt

    R_PG_Timer_StopModule_CMT_U0();        // Stop the CMT unit 0
}
```

5.12 Watchdog Timer (WDT)

5.12.1 R_PG_Timer_Start_WDT

Definition: bool R_PG_Timer_Start_WDT (void)

Description: Set up the WDT and start the count operation

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Timer_ WDT.c

RPDL function: R_WDT_Create

Details:

- Makes initial settings of WDT and starts the count operation.
  When the WDT is set to interval timer mode, the interval timer interrupt is set in this function. When the name of the interrupt notification function has been specified in the GUI, if an interrupt occurs in the CPU, the function having the specified name will be called. Create the interrupt notification function as follows:
  void <*name of the interrupt notification function*> (void)
  For the interrupt notification function, note the contents of 5.21, Notes on Notification Functions.

Example: A case where the setting is made as follows.

- The WDT has been set to interval timer mode.
- WdtIntFunc has been specified as a interval timer interrupt notification function name.

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
      R_PG_Clock_Set();    //Set up the clocks
      R_PG_Timer_Start_WDT();    //Set up the WDT and start the count operation
}

void WdtIntFunc(void)
{
      //Processing when the interval timer interrupt occurs
}
```

5.12.2 R_PG_Timer_HaltCount_WDT

Definition bool R_PG_Timer_HaltCount_WDT (void)

Description Stop the count operation

Parameter None

Return value

true	Halting succeeded
false	Halting failed

File for output R_PG_Timer_ WDT.c

RPDL function R_WDT_Control

Details
- Stops the WDT count operation.
- Call R_PG_Timer_Start_WDT to resume the count operation.

Example A case where the setting is made as follows.
- The WDT has been set to interval timer mode.
- WdtIntFunc has been specified as a interval timer interrupt notification function name.

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
      R_PG_Clock_Set();    //Set up the clocks
      R_PG_Timer_Start_WDT();    //Set up the WDT and start the count operation
}

void WdtIntFunc(void)
{
      R_PG_Timer_HaltCount_WDT();      //Halt the WDT count operation

      //Processing when the interval timer interrupt occurs

      R_PG_Timer_Start_WDT();    //Set up the WDT and start the count operation
}
```

5.12.3 R_PG_Timer_ResetCounter_WDT

Definition bool R_PG_Timer_ResetCounter_WDT(void)

Description Reset the counter of WDT

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_Timer_ WDT.c

RPDL function R_WDT_Control

Details

- Resets the counter of WDT

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func1(void)
{
      R_PG_Clock_Set();    //Set up the clocks
      R_PG_Timer_Start_WDT();    //Set up the WDT and start the count operation
}

void func2(void)
{
      R_PG_Timer_ResetCounter_WDT();      //Reset the WDT counter
}
```

5.12.4 R_PG_Timer_ClearOverflowFlag_WDT

Definition: bool R_PG_Timer_ClearOverflowFlag_WDT (bool* ov)

Description: Reset the counter of WDT

Parameter

bool* ov	The address of the storage area for the overflow flag

Return value

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Timer_ WDT.c

RPDL function: R_WDT_Read

Details
- This function acquires the counter overflow flags and clears.
- Specify 0 for a parameter if the flag is not required.

Example

A case where the setting is made as follows.
- The WDT has been set to interval timer mode.
- The priority level of interval timer interrupt has been set to 0.

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

bool ov;

void func(void)
{
    R_PG_Clock_Set();    //Set up the clocks
    R_PG_Timer_Start_WDT();    //Set up the WDT and start the count operation

    do{
        R_PG_Timer_ClearOverflowFlag_WDT( &ov ); //Get the overflow flag
    }while( !ov );

    /Processing when the interval timer interrupt occurs
}
```

5.13 Independent Watchdog Timer (IWDT)

5.13.1 R_PG_Timer_Set_IWDT

Definition: bool R_PG_Timer_Set_IWDT (void)

Description: Set up the IWDT

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Timer_IWDT.c

RPDL function: R_IWDT_Set

Details:

- Sets up the IWDT.
- The IWDT count operation starts by counter refresh.
  R_PG_Timer_RefreshCounter_IWDT can be used to refresh the counter

Example:

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

uint8_t output_val;

void func1(void)
{
      //Set up the clocks
      R_PG_Clock_Set();

      //Set up the IWDT
      R_PG_Timer_Set_WDT();

      //Start the count operation by refreshing the counter
      R_PG_Timer_RefreshCounter_IWDT();
}

void func2(void)
{
      //Refresh the counter
      R_PG_Timer_RefreshCounter_IWDT();
}
```

5.13.2 R_PG_Timer_RefreshCounter_IWDT

Definition: bool R_PG_Timer_RefreshCounter_IWDT (void)

Description: Refresh the counter

Parameter: None

Return value:

true	Refreshing succeeded
false	Refreshing failed

File for output: R_PG_Timer_IWDT.c

RPDL function: R_IWDT_Control

Details:

- Refreshes the IWDT counter
- To start the count operation, call this function after setting up IWDT by R_PG_Timer_Set_IWDT.
- After starting the count operation, call this function to clear the counter before the counter underflow.

Example: Refer to the example of R_PG_Timer_Set_IWDT

5.13.3 R_PG_Timer_GetCounterValue_IWDT

Definition: bool R_PG_Timer_GetCounterValue_IWDT(uint16_t * counter_val)

Description: Acquire the IWDT counter value

Parameter

uint16_t * counter_val	The address of the storage area for the IWDT counter value

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output: R_PG_Timer_IWDT.c

RPDL function: R_IWDT_Read

Details

- Acquires the IWDT counter value.
- The underflow flag shall be cleared in this function.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

uint16_t counter_val;

void func1(void)
{
    //Set up the clocks
    R_PG_Clock_Set();

    //Set up the IWDT
    R_PG_Timer_Set_WDT();

    //Start the count operation by refreshing the counter
    R_PG_Timer_RefreshCounter_IWDT();
}

void func2(void)
{
    R_PG_Timer_GetCounterValue_IWDT( &counter_val );

    if( counter_val < 0x1000){

        //Refresh the counter
        R_PG_Timer_RefreshCounter_IWDT();
    }
}
```

5.13.4 R_PG_Timer_ClearUnderflowFlag_IWDT

Definition

bool R_PG_Timer_ClearUnderflowFlag_IWDT(bool * un)

Description

Acquire and clear the underflow flag

Parameter

bool * un	The address of the storage area for the underflow flag

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output

R_PG_Timer_IWDT.c

RPDL function

R_IWDT_Read

Details

- Acquires and clears the underflow flag.
- Specify 0 for a parameter if the flag is not required.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

bool un;

void func(void)
{
    R_PG_Timer_ClearUnderflowFlag_IWDT ( &un );
    if(un){
        // Processing after a reset caused by a couter underflow
    }
}
```

5.14 Serial Communications Interface (SCIb)

5.14.1 R_PG_SCI_Set_C<channel number>

Definition: bool R_PG_SCI_Set_C<*channel number*> (void)
<*channel number*>: 0 to 2

Description: Set up a SCI channel

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_SCI_C<*channel number*>.c
<*channel number*>: 0 to 2

RPDL function: R_SCI_Create, R_SCI_Set

Details:

- Releases a SCI channel from the module-stop state, makes initial settings, and the direction (input or output) and input buffer for the pin to be used is set.
- Function R_PG_Clock_Set must be called before calling this function.
- When the name of the notification function has been specified in the GUI, if corresponding event occurs, the function having the specified name will be called. Create the notification function as follows:
  void <*name of the notification function*> (void)
  For the notification function, note the contents of 5.21, Notes on Notification Functions.

Example: SCI0 has been set in the GUI.

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

void func(void)
{
      R_PG_Clock_Set();              //The clock-generation circuit has to be set first.
      R_PG_SCI_Set_C0();          //Set up SCI0.
}
```

5.14.2 R_PG_SCI_StartSending_C<*channel number*>

Definition

bool R_PG_SCI_StartSending_C<*channel number*> (uint8_t * data, uint16_t count)

<*channel number*>: 0 to 2

Description

Start the data transmission

Conditions for output

- The function of transmission is selected for a SCI channel in GUI.
- "Notify the transmission completion of all data by function call" is selected as the data transmission method in GUI.

Parameter

uint8_t * data	The start address of the data to be sent.
uint16_t count	The number of the data to be sent. Set this to 0 if the transmit data is a character string (ending with a null character).

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_SCI_C<*channel number*>.c

<*channel number*>: 0 to 2

RPDL function

R_SCI_Send

Details

- This function starts the data transmission.
- This function is generated when "Notify the transmission completion of all data by function call" is selected as the data transmission method in GUI. This function returns immediately and the notification function having the specified name will be called when the last byte has been sent.

  Create the notification function as follows:

  void <*name of the notification function*> (void)

  For the notification function, note the contents of 5.21, Notes on Notification Functions.
- The number of transmitted data can be acquired by R_PG_SCI_GetSentDataCount_C<*channel number*>. The transmission can be terminated by calling R_PG_SCI_StopCommunication_C<*channel number*> before all bytes have been sent.
- The count of transmitted characters will loop back to 0 if 65536 characters are sent.

Example

SCI0 has been set as transmitter in the GUI.

Sci0TrFunc was specified as the name of the transmit end notification function in the GUI.

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

uint8_t data[255];

void func(void)
{
    R_PG_Clock_Set();            //The clock-generation circuit has to be set first.
    R_PG_SCI_Set_C0();           //Set up SCI0.
    R_PG_SCI_StartSending_C0(data, 255);          //Send 255 bytes of binary data.
}

//Transmit end notification function that called when all bytes have been sent
void Sci0TrFunc(void)
{
    //Shut down the SCI0
    R_PG_SCI_StopModule_C0();
}
```

5.14.3 R_PG_SCI_SendAllData_C<*channel number*>

Definition

bool R_PG_SCI_SendAllData_C<*channel number*> (uint8_t * data, uint16_t count)

<*channel number*>: 0 to 2

Description

Transmit all data

Conditions for output

- The function of transmission is selected for a SCI channel in GUI.
- Other than "Notify the transmission completion of all data by function call" is selected as the data transmission method in GUI.

Parameter

uint8_t * data	The start address of the data to be sent.
uint16_t count	The number of the data to be sent. Set this to 0 if the transmit data is a character string (ending with a null character).

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_SCI_C<*channel number*>.c

<*channel number*>: 0 to 2

RPDL function

R_SCI_Send

Details

- This function transmits all data.
- This function is generated when other than "Notify the transmission completion of all data by function call" is selected as the transmission method in GUI. This function waits until the last byte has been sent.
- The count of transmitted characters will loop back to 0 if 65536 characters are sent.

Example

SCI0 has been set as transmitter in the GUI.

"Wait at the transmission function until the last byte has been transmitted" is selected as the transmission method in GUI.

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

uint8_t data[255];

void func(void)
{
    R_PG_Clock_Set();              //The clock-generation circuit has to be set first.
    R_PG_SCI_Set_C0();             //Set up SCI0.
    R_PG_SCI_SendAllData_C0(data, 255);          //Send 255 bytes of binary data.
    R_PG_SCI_StopModule_C0();          //Shut down the SCI0
}
```

5.14.4 R_PG_SCI_GetSentDataCount_C<*channel number*>

Definition

bool R_PG_SCI_GetSentDataCount_C<*channel number*> (uint16_t * count)

<*channel number*>: 0 to 2

Description

Acquire the number of transmitted data

Conditions for output

The function of transmission is selected for a SCI channel and "Notify the transmission completion of all data by function call" is selected as the data transmission method in GUI.

Parameter

uint16_t * count	The storage location for the number of bytes that have been transmitted in the current transmission.

Return value

true	Acquisition of the data count succeeded
false	Acquisition of the data count failed

File for output

R_PG_SCI_C<*channel number*>.c

<*channel number*>: 0 to 2

RPDL function

R_SCI_GetStatus

Details

- When "Notify the transmission completion of all data by function call" is selected as the data transmission method in GUI, the number of transmitted data can be acquired by calling this function.

Example

SCI0 has been set as transmitter in the GUI.

Sci0TrFunc was specified as the name of the transmit end notification function in the GUI.

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

uint8_t data[255];

void func(void)
{
    R_PG_Clock_Set();           //The clock-generation circuit has to be set first.
    R_PG_SCI_Set_C0();          //Set up SCI0.
    R_PG_SCI_Send_C0(data, 255);        //Send 255 bytes of binary data.
}

//The transmit end notification function that called when all bytes have been sent
void Sci0TrFunc(void)
{
    //Shut down the SCI0
    R_PG_SCI_StopModule_C0();
}

//The function to check the number of transmitted data and terminate the transmission
void func_terminate_SCI(void)
{
    uint16_t count;
    // Acquire the number of transmitted data
    R_PG_SCI_GetSentDataCount_C0(&count);

    if( count > 32 ){
        R_PG_SCI_StopCommunication_C0();    //Terminate the transmission
    }
}
```

5.14.5 R_PG_SCI_StartReceiving_C<*channel number*>

Definition

bool R_PG_SCI_StartReceiving_C<*channel number*> (uint8_t * data, uint16_t count)

<*channel number*>: 0 to 2

Description

Start the data reception

Conditions for output

- The function of reception is selected for a SCI channel in GUI
- "Notify the reception completion of all data by function call" is selected as the data reception method in GUI

Parameter

uint8_t * data	The start address of the storage area for the expected data.
uint16_t count	The number of the data to be received.

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_SCI_C<*channel number*>.c

<*channel number*>: 0 to 2

RPDL function

R_SCI_Receive

Details

- This function starts the data reception.
- This function is generated when "Notify the reception completion of all data by function call" is selected as the data reception method in GUI. This function returns immediately and the notification function having the specified name will be called when the last byte has been received.

  Create the notification function as follows:

  void <*name of the notification function*> (void)

  For the notification function, note the contents of 5.21, Notes on Notification Functions.
- The number of received data can be acquired by R_PG_SCI_GetReceivedDataCount_C<*channel number*>. The reception can be terminated by calling R_PG_SCI_StopReceiving_C<*channel number*> before all bytes have been received.
- The maximum number of characters to be received is 65535.

Example

• SCI0 has been set as receiver in the GUI.

• Sci0ReFunc was specified as the name of the receive end notification function in the GUI.

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

uint8_t data[255];

void func(void)
{
    R_PG_Clock_Set();              //The clock-generation circuit has to be set first.
    R_PG_SCI_Set_C0();             //Set up SCI0.
    R_PG_SCI_StartReceiving_C0(data, 255);      //Receive 255 bytes of binary data.
}

//Receive end notification function that called when all bytes have been received
void Sci0ReFunc(void)
{
    //Shut down the SCI0
    R_PG_SCI_StopModule_C0();
}
```

5.14.6 R_PG_SCI_ReceiveAllData_C<*channel number*>

Definition

bool R_PG_SCI_ReceiveAllData_C<*channel number*> (uint8_t * data, uint16_t count)

<*channel number*>: 0 to 2

Description

Receive all data

Conditions for output

- The function of reception is selected for a SCI channel in GUI.
- Other than "Notify the reception completion of all data by function call" is selected as the data reception method in GUI

Parameter

uint8_t * data	The start address of the storage area for the expected data.
uint16_t count	The number of the data to be received.

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_SCI_C<*channel number*>.c

<*channel number*>: 0 to 2

RPDL function

R_SCI_Receive

Details

- This function receives all data.
- This function is generated when other than "Notify the reception completion of all data by function call" is selected as the data reception method in GUI. This function waits until the last byte has been received.
- The maximum number of characters to be received is 65535.

Example

SCI0 has been set as receiver in the GUI.

"Wait at the reception function until all data has been transmitted" is selected as the reception method in GUI.

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

uint8_t data[255];

void func(void)
{
    R_PG_Clock_Set();              //The clock-generation circuit has to be set first.
    R_PG_SCI_Set_C0();             //Set up SCI0.
    R_PG_SCI_ReceiveAllData_C0(data, 255);          //Receive 255 bytes of binary
data.
    R_PG_SCI_StopModule_C0();          //Shut down the SCI0
}
```

5.14.7 R_PG_SCI_StopCommunication_C<*channel number*>

Definition R_PG_SCI_StopCommunication_C<*channel number*> (void)
<*channel number*>: 0 to 2

Description Stop transmission and reception of serial data

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_SCI_C<*channel number*>.c
<*channel number*>: 0 to 2

RPDL function R_SCI_Control

Details
- This function stops data transmission and reception.
- When "Notify the transmission completion of all data by function call" is selected as the data transmission method in GUI, the reception can be terminated by calling this function before the number of bytes specified at R_PG_SCI_StartSending_C<*channel number*> have been received.
- When "Notify the reception completion of all data by function call" is selected as the data reception method in GUI, the reception can be terminated by calling this function before the number of bytes specified at R_PG_SCI_StartReceiving_C<*channel number*> have been received.

Example
SCI0 has been set as receiver in the GUI.
Sci0ReFunc was specified as the name of the receive end notification function in the GUI.

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

uint8_t data[255];

void func(void)
{
    R_PG_Clock_Set();              //The clock-generation circuit has to be set first.
    R_PG_SCI_Set_C0();             //Set up SCI0.
    R_PG_SCI_StartReceiving_C0(data, 255);          //Send 255 bytes of binary data.
}

//The receive end notification function that called when all bytes have been received.
void Sci0ReFunc(void)
{
    //Shut down the SCI0
    R_PG_SCI_StopModule_C0();
}

//The function to check the number of received data and terminate the reception
void func_terminate_SCI(void)
{
    uint8_t count;
    //Acquire the number of received data
    R_PG_SCI_GetReceivedDataCount_C0(&count);
    if( count > 32 ){
        R_PG_SCI_StopCommunication_C0();    //Terminate the reception
    }
}
```

5.14.8 R_PG_SCI_GetReceivedDataCount_C<*channel number*>

Definition

bool R_PG_SCI_GetReceivedDataCount_C<*channel number*> (uint16_t * count)

<*channel number*>: 0 to 2

Description

Acquire the number of received data

Conditions for output

The function of reception is selected for a SCI channel and "Notify the reception completion of all data by function call" is selected as the data reception method in GUI.

Parameter

uint16_t * count	The storage location for the number of bytes that have been received in the current reception process.

Return value

true	Acquisition of the data count succeeded
false	Acquisition of the data count failed

File for output

R_PG_SCI_C<*channel number*>.c

<*channel number*>: 0 to 2

RPDL function

R_SCI_GetStatus

Details

- When " Notify the reception completion of all data by function call " is selected as the receive end notification in GUI, the number of received data can be acquired by calling this function.

Example

SCI0 has been set as receiver in the GUI.

Sci0ReFunc was specified as the name of the receive end notification function in the GUI.

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

uint8_t data[255];

void func(void)
{
    R_PG_Clock_Set();           //The clock-generation circuit has to be set first.
    R_PG_SCI_Set_C0();          //Set up SCI0.
    R_PG_SCI_Receive_C0(data, 255);         //Send 255 bytes of binary data.
}

//The receive end notification function that called when all bytes have been received.
void Sci0ReFunc(void)
{
    //Shut down the SCI0
    R_PG_SCI_StopModule_C0();
}

//The function to check the number of received data and terminate the reception
void func_terminate_SCI(void)
{
    uint16_t count;
    //Acquire the number of received data
    R_PG_SCI_GetReceivedDataCount_C0(&count);
    if( count > 32 ){
        R_PG_SCI_StopReceiving_C0();    //Terminate the reception
    }
}
```

5.14.9 R_PG_SCI_GetReceptionErrorFlag_C<*channel number*>

Definition

bool R_PG_SCI_GetReceptionErrorFlag_C<*channel number*>
(bool * parity, bool * framing, bool * overrun)
<*channel number*>: 0 to 2

Description

Get the serial reception error flag

Conditions for output

The function of reception is selected for a SCI channel

Parameter

bool * parity	The address of the storage area for the parity error flag
bool * framing	The address of the storage area for the framing error flag
bool * overrun	The address of the storage area for the overrun error flag

Return value

true	Acquisition of the flags succeeded
false	Acquisition of the flags failed

File for output

R_PG_SCI_C<*channel number*>.c
<*channel number*>: 0 to 2

RPDL function

R_SCI_GetStatus

Details

- This function acquires the reception error flags.
- Specify the address of storage area for the flags to be acquired.
- Specify 0 for a flag that is not required.
- The flags of detected error will be set to 1.

Example

SCI0 has been set as receiver in the GUI.
Sci0ReFunc was specified as the name of the receive end notification function in the GUI.

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

uint8_t data[255];

void func(void)
{
      R_PG_Clock_Set();              //The clock-generation circuit has to be set first.
      R_PG_SCI_Set_C0();             //Set up SCI0.
      R_PG_SCI_Receive_C0(data, 1);          //Send 1bytes of binary data.
}

//The receive end notification function that called when all bytes have been received.
void Sci0ReFunc(void)
{
      // Acquire the reception error flags
      R_PG_SCI_GetReceptionErrorFlag_C0( &parity, &framing, & overrun );
}
```

5.14.10 R_PG_SCI_GetTransmitStatus_C<*channel number*>

Definition

bool R_PG_SCI_GetTransmitStatus_C<*channel number*> (bool * complete)

<*channel number*>: 0 to 2

Description

Get the state of transmission

Conditions for output

The function of transmission is selected for a SCI channel

Parameter

bool * complete	The address of the storage area for the transmission completion flag (0: Being transmitted 1:Complete)

Return value

true	Acquisition of the transmission status succeeded
false	Acquisition of the transmission status failed

File for output

R_PG_SCI_C<*channel number*>.c

<*channel number*>: 0 to 2

RPDL function

R_SCI_GetStatus

Details

- This function acquires the state of transmission.

Example

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

bool complete;

void func(void)
{
      //Get the state of transmission
      R_PG_SCI_GetTransmitStatus_C0( &complete );
}
```

5.14.11 R_PG_SCI_SendTargetStationID_C<*channel number*>

Definition

bool R_PG_SCI_SendTargetStationID_C<*channel number*> (uint8_t id)

<*channel number*>: 0 to 2

Description

Transmits the ID code of the receiving station

Conditions for output

- The function of transmission is selected for a SCI channel
- The multi-processer communications function is enabled in the asynchronous serial communication mode

Parameter

uint8_t id	The ID to be transmitted (0 to 255)

Return value

true	Transmission succeeded
false	Transmission failed

File for output

R_PG_SCI_C<*channel number*>.c

<*channel number*>: 0 to 2

RPDL function

R_SCI_Send

Details

- Generates an ID transmission cycle to transmit the ID code of the destination receiving station.
- This function waits until the ID transmission cycle has been completed.

Example

A case where the setting is made as follows.

- The function of transmission is selected for a SCI2 channel
- The multi-processer communications function is enabled in the asynchronous serial communication mode
- “Wait at the transmission function until all data has been transmitted” is selected as the data transmission method.

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

uint8_t data[10] = ”ABCDEFGHIJ”;

void func(void)
{
      R_PG_Clock_Set();              //The clock-generation circuit has to be set first.
      R_PG_SCI_Set_C2();             //Set up SCI2
      R_PG_SCI_SendTargetStationID_C2( 5 );          //Send ID code (ID:5)
      R_PG_SCI_SendAllData_C2( data, 10 );           //Send data
}
```

5.14.12 R_PG_SCI_ReceiveStationID_C<*channel number*>

Definition

bool R_PG_SCI_ReceiveStationID_C<*channel number*> (void)

<*channel number*>: 0 to 2

Description

Receives the ID code matches the ID of the receiving station itself

Conditions for output

- The function of reception is selected for a SCI channel
- The multi-processer communications function is enabled in the asynchronous serial communication mode

Parameter

None

Return value

true	Reception succeeded
false	Reception failed

File for output

R_PG_SCI_C<*channel number*>.c

<*channel number*>: 0 to 2

RPDL function

R_SCI_Receive

Details

- This function waits until the ID code matches the ID of the receiving station itself has been received.

Example

A case where the setting is made as follows.

- The function of reception is selected for a SCI0 channel
- The multi-processer communications function is enabled in the asynchronous serial communication mode
- "Notify the reception completion of all data by function call" is selected as the data reception method

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

uint8_t data[10];

void func(void)
{
    R_PG_Clock_Set();              //The clock-generation circuit has to be set first.
    R_PG_SCI_Set_C0();             //Set up SCI0
    R_PG_SCI_ReceiveStationID_C0();            //Wait an ID reception
    R_PG_SCI_StartReceiving_C0( data, 10 );            //Start receiving
}
```

5.14.13 R_PG_SCI_StopModule_C<*channel number*>

Definition bool R_PG_SCI_StopModule_C<*channel number*> (void)
<*channel number*>: 0 to 2

Description Shut down a SCI channel

Parameter None

Return value

true	Shutting down succeeded
false	Shutting down failed

File for output R_PG_SCI_C<*channel number*>.c
<*channel number*>: 0 to 2

RPDL function R_SCI_Destroy

Details
- Stops a SCI channel and places it in the module-stop state.

Example A case where the setting is made as follows.
- SCI0 has been set as receptor in the GUI.
- "Wait at the reception function until all data has been received" is selected as the data reception method instead of specifying the receive end notification function name in GUI.

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

uint8_t data[255];

void func(void)
{
    R_PG_Clock_Set();              //The clock-generation circuit has to be set first.
    R_PG_SCI_Set_C0();             //Set up SCI0.
    R_PG_SCI_Receive_C0(data, 255);    //Receive 255 bytes of binary data.
    R_PG_SCI_StopModule_C0();      //Shut down the SCI0
}
```

5.14.14 R_PG_SCI_ControlClockOutput_C<*channel number*>

Definition bool R_PG_SCI_ControlClockOutput_C<*channel number*> (bool output_enable)
<*channel number*>: 0 to 2

Description Control the SCKn pin output

Conditions for output
- The mode is selected for the smart card interface mode
- The GSM mode of the smart card interface mode is enabled.
  The SCKn pinfunction is selected output fixed low or high.

Parameter

bool output_enable	SCKn pin output (1:clock output 0:output fixed)

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_SCI_C<*channel number*>.c
<*channel number*>: 0 to 2

RPDL function R_SCI_Control

Details
-

Example A case where the setting is made as follows.
-
-
-

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

void func(void)
{
      R_PG_Clock_Set();              //The clock-generation circuit has to be set first.
      R_PG_SCI_Set_C0();             //Set up SCI0
      R_PG_SCI_ControlClockOutput_C0( true );    //
}
```

5.15 CRC Calculator (CRC)

5.15.1 R_PG_CRC_Set

Definition: bool R_PG_CRC_Set(void)

Description: Set up CRC calculator

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_CRC.c

RPDL function: R_CRC_Create

Details:

- Releases the CRC calculator from the module-stop state, makes initial settings.

Example:

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

uint16_t result;

void func(void)
{
    R_PG_CRC_Set();    //Set up the CRC calculator
    R_PG_CRC_InputData(0xf0);    // Write the payload data
    R_PG_CRC_InputData(0x8f);    // Write the first half of the CRC checksum
    R_PG_CRC_InputData(0x7f);    // Write the second half of the CRC checksum
    R_PG_CRC_GetResult (&result);    // Read the CRC calculation result
    R_PG_CRC_StopModule();    // Shutdown the CRC unit
}
```

5.15.2 R_PG_CRC_InputData

Definition bool R_PG_CRC_InputData (uint8_t data)

Description Input a data to CRC calculator

Parameter

uint8_t data	The data to be used for the calculation

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_CRC.c

RPDL function R_CRC_Write

Details

- This function writes the data into the CRC calculation register

Example Refer to the example of R_PG_CRC_Set.

5.15.3 R_PG_CRC_GetResult

Definition: bool R_PG_CRC_GetResult (uint16_t * result)

Description: Get the the result of calculation

Parameter

uint16_t * result	The address of the location where the result shall be stored.

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output: R_PG_CRC.c

RPDL function: R_CRC_Read

Details

- This function acquires the the result of calculation

Example: Refer to the example of R_PG_CRC_Set.

5.15.4 R_PG_CRC_StopModule

Definition bool R_PG_CRC_StopModule(void)

Description Shut down CRC calculator

Parameter None

Return value

true	Shutting down succeeded
false	Shutting down failed

File for output R_PG_CRC.c

RPDL function R_CRC_Destroy

Details
- Stops the CRC calculator and places it in the module-stop state.

Example Refer to the example of R_PG_CRC_Set.

5.16 I2C Bus Interface (RIIC)

5.16.1 R_PG_I2C_Set_C<*channel number*>

Definition: bool R_PG_I2C_Set_C<*channel number*> (void)
<*channel number*>: 0

Description: Set up an I2C bus interface channel

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_I2C_C<*channel number*>.c
<*channel number*>: 0

RPDL function: R_IIC_Create

Details:

- Releases an I2C bus interface channel from the module-stop state, makes initial settings, and the direction (input or output) and input buffer for the pin to be used is set. Function R_PG_Clock_Set must be called before calling this function.

Example: RIIC0 has been set in the GUI.

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

void func(void)
{
    R_PG_Clock_Set();          //The clock-generation circuit has to be set first
    R_PG_I2C_Set_C0();         //Set up RIIC0
}
```

5.16.2 R_PG_I2C_MasterReceive_C<*channel number*>

Definition

bool R_PG_I2C_MasterReceive_C<*channel number*>
(bool addr_10bit, uint16_t slave, uint8_t* data, uint16_t count)
<*channel number*>: 0

Description

Master data reception

Conditions for output

The function of master is selected for an I2C bus interface channel in GUI.

Parameter

bool addr_10bit	Target slave address format (0:7bit 1:10bit)
uint16_t slave	Target slave address
uint8_t* data	The start address of the storage area for the expected data.
uint16_t count	The number of the data to be received.

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_I2C_C<*channel number*>.c
<*channel number*>: 0

RPDL function

R_IIC_MasterReceive

Details

- This function reads data from slave module. The stop condition is generated when the specified number of data has been received and reception completes.
- If "Wait at the reception function until all data has been transmitted" is selected as the master reception method in GUI, this function waits until the last byte has been received.
- If "Notify the reception completion of all data by function call" is selected as the master reception method in GUI, this function returns immediately and the notification function having the specified name will be called when the last byte has been received.
  Create the notification function as follows:
  void <*name of the notification function*> (void)
  For the notification function, note the contents of 5.21, Notes on Notification Functions.
- A Start condition will be generated automatically. If the previous transfer did not issue a stop condition, a repeated start condition will be generated.
- In the 7-bit address mode, [8:1] of specified slave address value will be output. In 10-bit address mode, [10:1] of specified slave address will be output.
- The number of received data can be acquired by R_PG_I2C_GetReceivedDataCount_C <*channel number*>.
- When using 10-bit address mode, select other than [Notify the reception completion of all data by function call] for master reception method in the GUI.

Example A case where the setting is made as follows.

- The function of master is selected for a RIIC0
- "Wait at the reception function until all data has been transmitted" is selected as the master reception method

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

// The storage area for the received data
uint8_t iic_data[10];

void func(void)
{
      //The clock-generation circuit has to be set first
      R_PG_Clock_Set();

      //Set up RIIC0
      R_PG_I2C_Set_C0();

      //Master reception
     R_PG_I2C_MasterReceive_C0(
          0,      //Slave address format
          6,      //Slave address
          iic_data,      // The start address of the storage area for the received data
          10      // The number of the data to be received
     );

      //Stop RIIC0
     R_PG_I2C_StopModule_C0();
}
```

5.16.3 R_PG_I2C_MasterReceiveLast_C<*channel number*>

Definition

bool R_PG_I2C_MasterReceiveLast_C< *channel number* >
(uint8_t* data)
< *channel number* >: 0,1

Description

Complete a master reception process

Conditions for output

- The function of master is selected for an I2C bus interface channel in GUI.
- Select DTC transfer as a master reception method

Parameter

uint8_t* data	The address of the storage area for the expected data.

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_I2C_C<*channel number*>.c
<*channel number*>: 0

RPDL function

R_IIC_MasterReceiveLast

Details

- This function is genetarted when [Transfer the received serial data by DTC] is selected as a master reception method.
- In the master reception process that has used the DTC transfer, NACK and stop
- condition will be issued by calling this function and the reception process will be terminated.
- To complete reception process when the DTC transfer completes, call this function from DTC interrupt notification function.
- Extra 1 byte is acquired from the receive data register in this function.
- The events that has been detected during the reception process or the received data count can be acquired by calling R_PG_I2C_GetEvent_Cn or R_PG_I2C_GetReceivedDataCount_Cn.

Example A case where the setting is made as follows.

- "Transfer the received serial data by DTC" is selected as the master reception method in RIIC0 setting.
- DTC is set as follows

  Transfer request source : ICRXI0(receive data full interrupt of TIIC0)
  Transfer unit size : 1 byte
  Transfer count : Number of dtat to be received by RIIC0
  Source start address : Address of RIIC0 received data register
  Destination start address : Destination address of the data transfer
  Source address mode : Fix
  DMA interrupt notification fuction name : Dmac0IntFunc

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

void func1(void)
{
    //The clock-generation circuit has to be set first
    R_PG_Clock_Set();

    //Set up RIIC0
    R_PG_I2C_Set_C0();

    //Set up the DTC and make the transfer setting
    R_PG_DTC_Set();
    R_PG_DTC_Set_ICRXI0();

    //Activate the DTC
    R_PG_DTC_Activate();

    //Master reception
    //For DTC transfer, specify PDL_NO_PTR for the address of the storage area
    //For DTC transfer, specify 0 for the number of the data
    R_PG_PG_I2C_MasterReceive_C0(
        0,      //Slave address format
        6,      //Slave address
        PDL_NO_PTR, // The address of the storage area
        0       // The number of the data
    );
}

void func2(void)
{
    uint8_t data; //Storage area of extra data

    //Isse NACK and STOP condition and complete the reception
    R_PG_PG_I2C_MasterReceiveLast( &data );
}
```

5.16.4 R_PG_I2C_MasterSend_C<*channel number*>

Definition

bool R_PG_I2C_MasterSend_C<*channel number*>
(bool addr_10bit, uint16_t slave, uint8_t* data, uint16_t count)
<*channel number*>: 0

Description

Master data transmission

Conditions for output

The function of master is selected for an I2C bus interface channel in GUI.

Parameter

bool addr_10bit	Target slave address format (0:7bit 1:10bit)
uint16_t slave	Target slave address
uint8_t* data	The start address of the data to be sent
uint16_t count	The number of the data to be sent

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_I2C_C<*channel number*>.c
<*channel number*>: 0

RPDL function

R_IIC_MasterSend

Details

- This function sends data to the slave module. The stop condition is generated when the specified number of data has been transmitted and transmission completes.
- If "Wait at the transmission function until all data has been transmitted" is selected as the data transmission method in GUI, this function waits until the last byte has been transmitted or other events are detected.
- If "Notify the transmission completion of all data by function call" is selected as the data transmission method in GUI, this function returns immediately and the notification function having the specified name will be called when the last byte has been transmitted. Create the notification function as follows:
  void <*name of the notification function*> (void)
  For the notification function, note the contents of 5.21, Notes on Notification Functions.
- A Start condition will be generated automatically. If the previous transfer did not issue a stop condition, a repeated start condition will be generated.
- In the 7-bit address mode, [8:1] of specified slave address value will be output. In 10-bit address mode, [10:1] of specified slave address will be output.
- The number of transmitted data can be acquired by R_PG_I2C_GetSentDataCount_C <*channel number*>.
- When using 10-bit address mode, select other than [Notify the transmission completion of all data by function call] for master transmission method in the GUI.

Example A case where the setting is made as follows.

- The function of master is selected for a RIIC0
- "Wait at the transmission function until all data has been transmitted" is selected as the data transmission method

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

// The storage area for the data to be transmitted
uint8_t iic_data[10];

void func(void)
{
      //The clock-generation circuit has to be set first
      R_PG_Clock_Set();

      //Set up RIIC0
      R_PG_I2C_Set_C0();

      //Master transmission
     R_PG_I2C_MasterSend_C0(
          0,      //Slave address format
          6,      //Slave address
          iic_data,      // The start address of the storage area for the data to be transmitted
          10      // The number of the data to be transmitted
     );

      //Stop RIIC0
     R_PG_I2C_StopModule_C0();
}
```

5.16.5 R_PG_I2C_MasterSendWithoutStop_C<*channel number*>

Definition

bool R_PG_I2C_MasterSendWithoutStop_C<*channel number*>
(bool addr_10bit, uint16_t slave, uint8_t* data, uint16_t count)
<*channel number*>: 0

Description

Master data transmission (No stop condition)

Conditions for output

The function of master is selected for an I2C bus interface channel in GUI.

Parameter

bool addr_10bit	Target slave address format (0:7bit 1:10bit)
uint16_t slave	Target slave address
uint8_t* data	The start address of the data to be sent
uint16_t count	The number of the data to be sent

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_I2C_C<*channel number*>.c
<*channel number*>: 0

RPDL function

R_IIC_MasterSend

Details

- This function sends data to the slave module. The stop condition will not be generated. To generate a stop condition, call R_PG_I2C_GenerateStopCondition_C<*channel number*>.
- If "Wait at the transmission function until all data has been transmitted" is selected as the data transmission method in GUI, this function waits until the last byte has been transmitted or other events are detected.
- If "Notify the transmission completion of all data by function call" is selected as the data transmission method in GUI, this function returns immediately and the notification function having the specified name will be called when the last byte has been transmitted. Create the notification function as follows:
  void <*name of the notification function*> (void)
- For the notification function, note the contents of 5.21, Notes on Notification Functions.
- A Start condition will be generated automatically. If the previous transfer did not issue a stop condition, a repeated start condition will be generated.
- In the 7-bit address mode, [8:1] of specified slave address value will be output. In 10-bit address mode, [10:1] of specified slave address will be output.
- The number of transmitted data can be acquired by R_PG_I2C_GetSentDataCount_C<*channel number*>.
- When using 10-bit address mode, select other than [Notify the transmission completion of all data by function call] for master transmission method in the GUI.

Example A case where the setting is made as follows.

- The function of master is selected for a RIIC0
- "Notify the transmission completion of all data by function call" is selected as the data transmission method
- IIC0MasterTrFunc was specified as the name of the transmit end notification function

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

// The storage area for the data to be transmitted
uint8_t iic_data[10];

void func(void)
{
    //The clock-generation circuit has to be set first
    R_PG_Clock_Set();

    //Set up RIIC0
    R_PG_I2C_Set_C0();

    //Master transmission
    R_PG_I2C_MasterSendWithoutStop_C0(
        0,      //Slave address format
        6,      //Slave address
        iic_data,   // The start address of the storage area for the data to be transmitted
        10      // The number of the data to be transmitted
    );
}

void IIC0MasterTrFunc(void){

    //Generate stop condition
    R_PG_I2C_GenerateStopCondition_C0();

    //Stop RIIC0
    R_PG_I2C_StopModule_C0();
}
```

5.16.6 R_PG_I2C_GenerateStopCondition_C<*channel number*>

Definition
bool R_PG_I2C_GenerateStopCondition_C<*channel number*> (void)
<*channel number*>: 0

Description
Generate a stop condition

Conditions for output
The function of master is selected for an I2C bus interface channel in GUI.

Parameter
None

Return value

true	Setting was made correctly
false	Setting failed

File for output
R_PG_I2C_C<*channel number*>.c
<*channel number*>: 0

RPDL function
R_IIC_Control

Details
- This function generates a stop condition for the transmission started by R_PG_I2C_MasterSendWithoutStop_C<*channel number*>.

Example
Refer to the example of R_PG_I2C_MasterSendWithoutStop_C<*channel number*>

5.16.7 R_PG_I2C_GetBusState_C<*channel number*>

Definition

bool R_PG_I2C_GetBusState_C<*channel number*> (bool *busy)

<*channel number*>: 0

Description

Get the bus state

Conditions for output

The function of master is selected for an I2C bus interface channel in GUI.

Parameter

bool *busy	The address of the storage area for the bus busy detection flag Bus busy detection flag 0: The I2C bus is released (bus free state) 1: The I2C bus is occupied (bus busy state or in the bus free state)

Return value

true	Acquisition of the flag succeeded
false	Acquisition of the flag failed

File for output

R_PG_I2C_C<*channel number*>.c

<*channel number*>: 0

RPDL function

R_IIC_GetStatus

Details

- This function acquires the bus busy detection flag.

Example

RIIC0 has been set in the GUI.

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"
// The storage area for the data to be transmitted
uint8_t iic_data[10];
//Storage for bus busy detection flag
bool busy;
void func(void)
{
    //The clock-generation circuit has to be set first
    R_PG_Clock_Set();
    //Set up RIIC0
    R_PG_I2C_Set_C0();
    // Wait for the I2C bus to be free
    do{
        R_PG_I2C_GetBusState_C0( & busy );
    } while( busy );
    //Master transmission
    R_PG_I2C_MasterSend_C0(
        0,      //Slave address format
        6,      //Slave address
        iic_data,   // The start address of the storage area for the data to be transmitted
        10      // The number of the data to be transmitted
    );
```

5.16.8 R_PG_I2C_SlaveMonitor_C<*channel number*>

Definition

bool R_PG_I2C_SlaveMonitor_C<*channel number*> (uint8_t *data, uint16_t count)

<*channel number*>: 0

Description

Slave bus monitor

Conditions for output

The function of slave is selected for an I2C bus interface channel in GUI.

Parameter

uint8_t* data	The start address of the received data
uint16_t count	The number of the data to be received

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_I2C_C<*channel number*>.c

<*channel number*>: 0

RPDL function

R_IIC_SlaveMonitor

Details

- This function monitors the accesses from master modules.
- If "Notify the reception completion of all data, slave read request, or a stop condition detection by function call" is selected as the slave monitor method in GUI, this function returns immediately and the notification function having the specified name will be called when a read access from master module or a stop condition is detected. Create the notification function as follows:

  void <*name of the notification function*> (void)

  For the notification function, note the contents of 5.21, Notes on Notification Functions.
- If "Wait at the monitor function until reception completion, slave read request, or a stop condition detection" is selected as the slave monitor method in GUI, this function waits until a read access from master module or a stop condition is detected.
- The received data from a master module is stored in the storage area of specified address. Specify the number of data to not exceed the size of storage area. If the number of the data from the master module exceeds the specified number, NACK shall be generated.
- The transmit/receive mode can be acquired by calling R_PG_I2C_GetRW_C<*channel number*>. The data can be transmitted by calling R_PG_I2C_SlaveSend_C<*channel number*> to respond to a transmission (read) request from the master.
- Call R_PG_I2C_GetDetectedAddress_C<*channel number*> to acquire a detected slave address. Call R_PG_I2C_GetEvent_C<*channel number*> to acquire the detected events (e.g. a stop condition or a start condition).
- When using 10-bit address mode, select other than [Notify the transmission completion of all data, slave read request, or a stop condition detection by function call] for slave monitor method in the GUI.

Example A case where the setting is made as follows.

- The function of slave is selected for a RIIC0
- IIC0SlaveFunc was specified as the name of the slave monitor function

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"
// The storage area for the data to be received
uint8_t iic_data_re[10];
// The storage area for the data to be transmitted (slave address 0)
uint8_t iic_data_tr_0[10];
// The storage area for the data to be transmitted (slave address 1)
uint8_t iic_data_tr_1[10];
//Storage for bus busy detection flag
uint8_t busy;
void func(void)
{
    //The clock-generation circuit has to be set first
    R_PG_Clock_Set();
    //Set up RIIC0
    R_PG_I2C_Set_C0();
    // Slave monitor
    R_PG_I2C_SlaveMonitor_C0(
        iic_data_re,    // The start address of the storage area for the received data
        10      //The number of the data to be received
    );
}
void IIC0SlaveFunc (void)
{
    bool transmit, start, stop;
    bool addr0, addr1;
    //Get the detected events
    R_PG_I2C_GetEvent_C0(0, &stop, &start, 0, 0);
    //Get an access type
    R_PG_PG_I2C_GetTR_C0(&transmit);
    //Get a detected address
    R_PG_I2C_GetDetectedAddress_C0(&addr0, &addr1, 0, 0, 0, 0);
    if (start && transmit && address0) {
        //Transmits the data to the master module
        R_PG_I2C_SlaveSend_C(
            iic_data_tr_0,
            10
        );
    }
    else if (start && read && address1) {
        //Transmits the data to the master module
        R_PG_I2C_SlaveSend_C(
            iic_data_tr_1,
            10
        );
    }
}
```

5.16.9 R_PG_I2C_SlaveSend_C<*channel number*>

Definition

bool R_PG_I2C_SlaveSend_C<*channel number*> (uint8_t *data, uint16_t count)

<*channel number*>: 0

Description

Slave data transmission

Conditions for output

The function of slave is selected for an I2C bus interface channel in GUI.

Parameter

uint8_t* data	The start address of the data to be transmitted
uint16_t count	The number of the data to be transmitted

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_I2C_C<*channel number*>.c

<*channel number*>: 0

RPDL function

R_IIC_SlaveSend

Details

- This function transmits the data to the master module.
- If the master requires more data than is supplied, this function shall loop back to the start of the data.

Example

Refer to the example of R_PG_I2C_SlaveMonitor_C<*channel number*>

5.16.10 R_PG_I2C_GetDetectedAddress_C<*channel number*>

Definition

bool R_PG_I2C_GetDetectedAddress_C<*channel number*>
(bool *addr0, bool *addr1, bool *addr2, bool *general, bool *device, bool *host)
<*channel number*>: 0

Description

Get the detected address

Conditions for output

The function of slave is selected for an I2C bus interface channel in GUI.

Parameter

bool *addr0	The address of the storage area for slave address 0 detection flag
bool *addr1	The address of the storage area for slave address1 detection flag
bool *addr2	The address of the storage area for slave address 2 detection flag
bool *general	The address of the storage area for general call address detection flag
bool *device	The address of the storage area for device-ID command detection flag
bool *host	The address of the storage area for host address detection flag

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output

R_PG_I2C_C<*channel number*>.c
<*channel number*>: 0

RPDL function

R_IIC_GetStatus

Details

- This function acquires the detected address.
- Specify the address of storage area for the flags to be acquired.
- Specify 0 for a flag that is not required.
- The flag of the detected address will be set to 1.

Example

Refer to the example of R_PG_I2C_SlaveMonitor_C<*channel number*>

5.16.11 R_PG_I2C_GetTR_C<*channel number*>

Definition

bool R_PG_I2C_GetTR_PG_C<*channel number*> (bool * transmit)

<*channel number*>: 0

Description

Get the transmit/receive mode

Conditions for output

The function of slave is selected for an I2C bus interface channel in GUI.

Parameter

bool * transmit	The address of the storage area for the transmit/receive mode flag Transmit/receive mode flag 0: Receive mode 1: Transmit mode

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output

R_PG_I2C_C<*channel number*>.c

<*channel number*>: 0

RPDL function

R_IIC_GetStatus

Details

- This function acquires the transmit/receive mode flag.

Example

Refer to the example of R_PG_I2C_SlaveMonitor_C<*channel number*>

5.16.12 R_PG_I2C_GetEvent_C<*channel number*>

Definition

bool R_PG_I2C_GetEvent_C<*channel number*>
(bool *nack, bool *stop, bool *start, bool *lost, bool *timeout)
<*channel number*>: 0

Description

Get the detected event

Parameter

bool *nack	The address of the storage area for a NACK detection flag
bool *stop	The address of the storage area for a stop condition detection flag
bool *start	The address of the storage area for a start condition detection flag
bool *lost	The address of the storage area for an arbitration lost
bool *timeout	The address of the storage area for a timeout detection

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output

R_PG_I2C_C<*channel number*>.c
<*channel number*>: 0

RPDL function

R_IIC_GetStatus

Details

- This function acquires the detected event.
- Specify 0 for a flag that is not required.
- The flags of the detected event will be set to 1.

Example

Refer to the example of R_PG_I2C_SlaveMonitor_C<*channel number*>

5.16.13 R_PG_I2C_GetReceivedDataCount_C<*channel number*>

Definition

bool R_PG_I2C_GetReceivedDataCount_C<*channel number*> (uint16_t *count)

<*channel number*>: 0

Description

Acquires the count of received data

Parameter

uint16_t *count	The address of the storage area for the number of bytes that have been received

Return value

true	Acquisition of the data count succeeded
false	Acquisition of the data count failed

File for output

R_PG_I2C_C<*channel number*>.c

<*channel number*>: 0

RPDL function

R_IIC_GetStatus

Details

- This function acquires the number of bytes that have been received in the current reception process.

Example

A case where the setting is made as follows.

- The function of master is selected for a RIIC0
- "Notify the reception completion of all data by function call" is selected as the master reception method

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"
// The storage area for the data to be received
uint8_t iic_data[256];
// The storage area for the number of received data
uint16_t count;
void func(void)
{
    //The clock-generation circuit has to be set first
    R_PG_Clock_Set();
    //Set up RIIC0
    R_PG_I2C_Set_C0();
    //Master receive
    R_PG_I2C_MasterReceive_C0(
        0,      //Slave address format
        6,      //Slave address
        iic_data,   // The address of storage area for the data to be received
        256     //The number of data to be received
    );
    //Wait until 64 bytes have been received
    do{
        R_PG_I2C_GetReceivedDataCount_C0( &count );
    } while( count < 64 );
}
```

5.16.14 R_PG_I2C_GetSentDataCount_C<*channel number*>

Definition

bool R_PG_I2C_GetSentDataCount_C<*channel number*> (uint16_t *count)

<*channel number*>: 0

Description

Acquires the count of transmitted data

Parameter

uint16_t *count	The address of the storage area for the number of bytes that have been transmitted

Return value

true	Acquisition of the data count succeeded
false	Acquisition of the data count failed

File for output

R_PG_I2C_C<*channel number*>.c

<*channel number*>: 0

RPDL function

R_IIC_GetStatus

Details

- This function acquires the number of bytes that have been transmitted in the current transmission process.

Example

A case where the setting is made as follows.

- The function of master is selected for a RIIC0
- "Notify the transmission completion of all data by function call" is selected as the data transmission method

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

// The storage area for the data to be transmitted
uint8_t iic_data[256];

// The storage area for the number of transmitted data
uint16_t count;

void func(void)
{
    //The clock-generation circuit has to be set first
    R_PG_Clock_Set();

    //Set up RIIC0
    R_PG_I2C_Set_C0();

    //Master send
    R_PG_I2C_MasterSend_C0(
        0,      //Slave address format
        6,      //Slave address
        iic_data,   // The address of storage area for the data to be transmitted
        256     //The number of data to be transmitted
    );

    //Wait until 64 bytes have been transmitted
    do{
        R_PG_I2C_GetSentDataCount_C0( &count );
    } while( count < 64 );
}
```

5.16.15 R_PG_I2C_Reset_C<*channel number*>

Definition

bool R_PG_I2C_Reset_C<*channel number*> (void)

<*channel number*>: 0

Description

Reset the bus

Parameter

None

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_I2C_C<*channel number*>.c

<*channel number*>: 0

RPDL function

R_IIC_Control

Details

- This function resets the module
- The settings of the module are preserved.

Example

A case where the setting is made as follows.

- The function of master is selected for a RIIC0
- "Notify the transmission completion of all data by function call" is selected as the data transmission method

  IIC0MasterTrFunc was specified as the name of the transmit end notification function

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"
// The storage area for the data to be transmitted
uint8_t iic_data[256];
void func(void)
{
    //The clock-generation circuit has to be set first
    R_PG_Clock_Set();
    //Set up RIIC0
    R_PG_I2C_Set_C0();
    //Master send
    R_PG_I2C_MasterSend_C0(
        0,      //Slave address format
        6,      //Slave address
        iic_data,   // The address of storage area for the data to be transmitted
        10      //The number of data to be transmitted
    );
}
void IIC0MasterTrFunc(void)
{
    if ( error ){
        R_PG_I2C_Reset_C0();
    }
}
```

5.16.16 R_PG_I2C_StopModule_C<*channel number*>

Definition bool R_PG_I2C_StopModule_C<*channel number*> (void)

<*channel number*>: 0

Description Shut down the I2C bus interface channel

Parameter None

Return value

true	Shutting down succeeded
false	Shutting down failed

File for output R_PG_I2C_C<*channel number*>.c

<*channel number*>: 0

RPDL function R_IIC_Destroy

Details

- Stops an I2C bus interface channel and places it in the module-stop state.

Example A case where the setting is made as follows.

- The function of master is selected for a RIIC0
- "Wait at the reception function until all data has been transmitted" is selected as the master reception method

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"
// The storage area for the data to be transmitted
uint8_t iic_data[256];
void func(void)
{
    //The clock-generation circuit has to be set first
    R_PG_Clock_Set();
    //Set up RIIC0
    R_PG_I2C_Set_C0();
    //Master receive
    R_PG_I2C_MasterReceive_C0(
        0,      //Slave address format
        6,      //Slave address
        iic_data,   // The address of storage area for the data to be received
        10      //The number of data to be received
    );
    //Stop the RIIC0
    R_PG_I2C_StopModule_C0();
}
```

5.17 Serial Peripheral Interface (RSPI)

5.17.1 R_PG_RSPI_Set_C<*channel number*>

Definition: bool R_PG_RSPI_Set_C<*channel number*> (void)
<*channel number*>: 0

Description: Set up a RSPI channel

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_RSPI_C<*channel number*>.c
<*channel number*>: 0

RPDL function: R_SPI_Create

Details:

- Releases a serial peripheral interface channel from the module-stop state, makes initial settings, and sets the pins to be used.
- Function R_PG_Clock_Set must be called before calling this function.
- The commands are not set in this function. To set the commands, call R_PG_RSPI_SetCommand_C<*channel number*>.

Example:

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
      R_PG_Clock_Set();       //Set up the clocks
      R_PG_RSPI_Set_C0();       //Set up RSPI0
      R_PG_RSPI_SetCommand_C0();       //Set commands
}
```

5.17.2 R_PG_RSPI_SetCommand_C<*channel number*>

Definition bool R_PG_RSPI_SetCommand_C<*channel number*> (void)
<*channel number*>: 0

Description Set commands

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_RSPI_C<*channel number*>.c
<*channel number*>: 0

RPDL function R_SPI_Command

Details
- Set RSPI commands registers.
- All commands set in GUI (maximum number of commands: 8) shall be set.

Example Refer to the example of R_PG_RSPI_Set_C<*channel number*>

5.17.3 R_PG_RSPI_StartTransfer_C<*channel number*>

Definition

Transmission and reception operations (Full-duplex synchronous serial communications)

bool R_PG_RSPI_StartTransfer_C<*channel number*>
(uint32_t * tx_start, uint32_t * rx_start, uint16_t sequence_loop_count)
<*channel number*>: 0

Serial communications consisting of only transmit operations

bool R_PG_RSPI_StartTransfer_C<*channel number*>
(uint32_t * tx_start, uint16_t sequence_loop_count)
<*channel number*>: 0

Description

Start the data transfer

Conditions for output

"Notify the transfer completion and the error detection by function call" has been selected as the transfer method.

Parameter

uint32_t * tx_start	The start address of the data to be transmitted.
uint32_t * rx_start	The start address of the storage area for the expected data.
uint16_t sequence_loop_count	The number of times that the command sequence will be executed

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_RSPI_C<*channel number*>.c
<*channel number*>: 0

RPDL function

R_SPI_Transfer

Details

- Starts the data transfer.
- This function is generated when "Notify the transfer completion and the error detection by function call" is selected as the data transfer method in GUI.
- This function returns immediately and the notification function having the specified name will be called when all commands are executed or error is detected.
  Create the notification function as follows:
  void <*name of the notification function*> (void)
  For the notification function, note the contents of 5.21, Notes on Notification Functions.

Example A case where the setting is made as follows.

- RSPI has been set to master mode
- “Notify the transfer completion and the error detection by function call” is selected as
- the transfer method
- rsi0_int_func is specified as a notification function name
- Number of commands: 1 Number of frames: 4
  Data lengh of command 0 is 8 bits

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

uint32_t tx_data[4] = { 0x11, 0x22, 0x33, 0x44 };
uint32_t rx_data[4] = { 0x00, 0x00, 0x00, 0x00 };
bool    over_run,    mode_fault,    parity_error;

void func(void)
{
      R_PG_Clock_Set();       //Set up the clocks
      R_PG_RSPI_Set_C0();       //Set up RSPI0
      R_PG_RSPI_SetCommand_C0();       //Set commands
      R_PG_RSPI_StartTransfer_C0( tx_data, rx_data, 1 ); //Transfe 4 frames * 8bits
}

void rsi0_int_func (void)
{
      R_PG_RSPI_GetError_C0(&over_run,&mode_fault,&parity_error); //Get error flags
      if( over_run || mode_fault || parity_error ){

            //Processing when an error is detected
      }
      R_PG_RSPI_StopModule_C0();
}
```

5.17.4 R_PG_RSPI_TransferAllData_C<*channel number*>

Definition

Transmission and reception operations (Full-duplex synchronous serial communications)

bool R_PG_RSPI_TransferAllData_C<*channel number*>
(uint32_t * tx_start, uint32_t * rx_start, uint16_t sequence_loop_count)
<*channel number*>: 0

Serial communications consisting of only transmit operations

bool R_PG_RSPI_TransferAllData_C<*channel number*>
(uint32_t * tx_start, uint16_t sequence_loop_count)
<*channel number*>: 0

The DTC transfer is selected for the transfer method

bool R_PG_RSPI_TransferAllData_C<*channel number*>
(uint16_t sequence_loop_count)
<*channel number*>: 0

Description

Transfer all data

Conditions for output

Other than “Notify the transfer completion and the error detection by function call” has been selected as the transfer method.

Parameter

uint32_t * tx_start	The start address of the data to be transmitted.
uint32_t * rx_start	The start address of the storage area for the expected data.
uint16_t sequence_loop_count	The number of times that the command sequence will be executed

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_RSPI_C<*channel number*>.c
<*channel number*>: 0

RPDL function

R_SPI_Transfer

Details

- Transfers all data.
- This function is generated when other than "Notify the transfer completion and the error detection by function call" is selected as the transmission method in GUI.
- This function waits until all commands are executed.

Example A case where the setting is made as follows.

- RSPI has been set to master mode.
- “Wait until transfer completion” is selected as the transfer method.
- Number of commands: 1 Number of frames: 4
- Data lengh of command 0 is 8 bits

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

uint32_t    tx_data[4] = { 0x11, 0x22, 0x33, 0x44 };
uint32_t    rx_data[4] = { 0x00, 0x00, 0x00, 0x00 };
bool    over_run,    mode_fault,    parity_error;

void func(void)
{
      R_PG_Clock_Set();       //Set up the clocks
      R_PG_RSPI_Set_C0();       //Set up RSPI0
      R_PG_RSPI_SetCommand_C0();       //Set commands
      R_PG_RSPI_TransferAllData_C0( tx_data, rx_data, 1 ); //Transfe 4 frames * 8bits

      R_PG_RSPI_GetError_C0(&over_run,&mode_fault,&parity_error); //Get error flags
      if( over_run || mode_fault || parity_error ){

            //Processing when an error is detected
      }
      R_PG_RSPI_StopModule_C0();
}
```

5.17.5 R_PG_RSPI_GetStatus_C<*channel number*>

Definition

bool R_PG_RSPI_GetStatus_C<*channel number*>
(bool * idle, bool * receive_full, bool * transmit_empty)
<*channel number*>: 0

Description

Acquire the transfer status

Parameter

bool * idle	The address of the storage area for the idle flag (0 : Idle state 1: Transfer state)
bool * receive_full	The address of the storage area for the receive buffer full flag (0 : Empty 1: Full)
bool * transmit_empty	The address of the storage area for the transmit buffer empty flag (0 : Full 1: Empty)

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output

R_PG_RSPI_C<*channel number*>.c
<*channel number*>: 0

RPDL function

R_SPI_GetStatus

Details

- Acquires the transfer status.
- Specify the address of storage area for the items to be acquired. Specify 0 for an item that is not required.
- The error flags (the overrun error flag, the mode fault error flag, and the parity error flag) are cleared in this function. Call R_PG_RSPI_GetError_C<*channel number*> to acquire the error flags before calling this function if needed.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

bool idle;

void func(void)
{
    do{
        //Get the id
        R_PG_RSPI_GetStatus_C0( & idle, 0, 0 );
    }while( idle );
}
```

5.17.6 R_PG_RSPI_GetError_C<*channel number*>

Definition

bool R_PG_RSPI_GetError_C<*channel number*>
(bool * over_run, bool * mode_fault, bool * parity_error)
<*channel number*>: 0

Description

Acquire the error flags

Parameter

bool * over_run	The address of the storage area for the overrun error flag
bool * mode_fault	The address of the storage area for the mode fault error flag
bool * parity_error	The address of the storage area for the parity error flag

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output

R_PG_RSPI_C<*channel number*>.c
<*channel number*>: 0

RPDL function

R_SPI_GetStatus

Details

- Acquires the error flags.
- Specify the address of storage area for the items to be acquired. Specify 0 for an item that is not required.
- The error flags shall be cleared in this function.

Example

Refer to the example of R_PG_RSPI_StartTransfer_C<*channel number*>, R_PG_RSPI_TransferAllData_C<*channel number*>, and R_PG_RSPI_GetCommandStatus_C<*channel number*>

5.17.7 R_PG_RSPI_GetCommandStatus_C<*channel number*>

Definition

bool R_PG_RSPI_GetCommandStatus_C<*channel number*>
(uint8_t * current_command, uint8_t * error_command)
<*channel number*>: 0

Description

Acquire the command status

Conditions for output

A RSPI channel has been set to the master mode

Parameter

uint8_t * current_command	The address of the storage area for the current command pointer value (0 to 7)
uint8_t * error_command	The address of the storage area for the value of command pointer when an error is detected (0 to 7)

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output

R_PG_RSPI_C<*channel number*>.c
<*channel number*>: 0

RPDL function

R_SPI_GetStatus

Details

- Acquires the current command pointer value (0 to 7) and the value of command pointer when an error is detected (0 to 7).
- Specify the address of storage area for the items to be acquired. Specify 0 for an item that is not required.
- The error flags (the overrun error flag, the mode fault error flag, and the parity error flag) are cleared in this function. Call R_PG_RSPI_GetError_C<*channel number*> to acquire the error flags before calling this function if needed.

Example

A case where the setting is made as follows.

- RSPI has been set to the master mode

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

bool over_run, mode_fault, parity_error;
uint8_t error_command;

void func(void)
{
    R_PG_RSPI_GetError_C0(&over_run,&mode_fault,&parity_error); //Get error flags
    if( over_run || mode_fault || parity_error ){
        R_PG_RSPI_GetCommandStatus_C0( &error_command );

        // Processing when an error is detected
    }
}
```

5.17.8 R_PG_RSPI_StopModule_C<*channel number*>

Definition: bool R_PG_RSPI_StopModule_C<*channel number*> (void)
<*channel number*>: 0

Description: Shut down a RSPI channel

Parameter: None

Return value:

true	Shutting down succeeded
false	Shutting down failed

File for output: R_PG_RSPI_C<*channel number*>.c
<*channel number*>: 0

RPDL function: R_SPI_Destroy

Details:
- Stops RSPI channel and places it in the module-stop state.

Example: Refer to the example of R_PG_RSPI_StartTransfer_C<*channel number*> and R_PG_RSPI_TransferAllData_C<*channel number*>.

5.17.9 R_PG_RSPI_LoopBack<*loopback mode*>_C<*channel number*>

Definition: bool R_PG_RSPI_LoopBack<*loopback mode*>_C<*channel number*> (void)
<*loopback mode*>: Direct, Reversed, Disable
<*channel number*>: 0

Description: Set loopback mode

Conditions for output: The loopback mode has been set

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_RSPI_C<*channel number*>.c
<*channel number*>: 0

RPDL function: R_SPI_Control

Details:

- Sets or disables RSPI pins to loopback mode.
- By calling R_PG_RSPI_LoopBackDirect_C<*channel number*>, the input path and output path for the shift register are connected. (transmit data = receive data)
- By calling R_PG_RSPI_LoopBackReversed _C<*channel number*>, the reversed input path and output path for the shift register are connected. (reversed transmit data = receive data)
- By calling R_PG_RSPI_LoopBackDisable_C<*channel number*>, the loopback mode is disabled.

Example:

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
      R_PG_RSPI_LoopBackDirect_C0();    //Set loopback mode
}
```

5.18 LIN Module (LIN)

5.18.1 R_PG_LIN_Set_LIN<*channel number*>

Definition

bool R_PG_LIN_Set_LIN<*channel number*> (void)

<*channel number*>: 0

Description

Set up a LIN module

Parameter

None

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_LIN_LIN<*channel number*>.c

<*channel number*>: 0

RPDL function

R_LIN_Create

Details

- Releases a LIN module from the module-stop state, makes initial settings, and sets the pins to be used.
- Sets up a LIN module in LIN operation mode.
- Interrupts of the LIN module are set by this function. When the name of the interrupt notification function has been specified in the GUI, if an interrupt occurs in the CPU, the function having the specified name will be called. Create the interrupt notification function as follows:

  void <*name of the interrupt notification function*> (void)

  For the interrupt notification function, note the contents of 5.21, Notes on Notification Functions.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
    //Initialize LIN module    (LIN module is set to LIN operation mode)
    R_PG_LIN_Set_LIN0();

    //Enter wake-up mode
    R_PG_LIN_EnterWakeUpMode_LIN0();

    //Wake-up transmission
    R_PG_LIN_WakeUpTransmit_LIN0();
}
```

5.18.2 R_PG_LIN_Transmit_LIN<*channel number*>

Definition

bool R_PG_LIN_Transmit_LIN<*channel number*>
(uint8_t id, uint8_t * send_data, uint8_t data_count, bool checksum_enhanced)
<*channel number*>: 0

Description

Transmit data

Parameter

uint8_t id	ID to be sent in the header
uint8_t * send_data	The start address of the data to be transmitted in the response field
uint8_t data_count	The number of the data to be sent
bool checksum_enhanced	Check sum select (0: Classic 1: Enhanced)

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_LIN_LIN<*channel number*>.c
<*channel number*>: 0

RPDL function

R_LIN_Transfer

Details

- Transmit the header and response.
- Check sum is automatically calculated and added to the response.
- Call R_PG_LIN_Set_LIN<*channel number*> before calling this function to set up the LIN module.
- This function needs to be called in LIN operation mode. To enter LIN operation mode from other modes, use R_PG_LIN_EnterOperationMode_LIN<*channel number*>.
- The status of transmission of the header and response can be acquired by R_PG_LIN_GetStatus_LIN<*channel number*>.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

// Storage area for the transmit data
uint8_t data_t[8];

// Storage area for the frame/wake-up transmit completion flag
bool frame_wakeup_transmission;

// Storage area for the error flag
bool error;

void func(void)
{
	//Initialize transmit data
	InitData();

	//Transmit header and response (ID:3	Number of data:8 Check sum:Classic)
	R_PG_LIN_Transmit_LIN0( 3, data_t, 8, 0 );
}

void Lin0IntFunc(void)
{
	//Get the frame/wake-up transmit completion flag and error flag
	R_PG_LIN_GetStatus_LIN0(
		& frame_wakeup_transmission,
		0,
		& error,
		0,
		0,
	);

	if( error ){
		//Error is detected
	}
	else if( frame_wakeup_transmission ){
		// Frame/wake-up transmission is completed
	}
}

void InitData(void)
{
	t data_t[0] = 0x12;
	t data_t[1] = 0x34;
	t data_t[2] = 0x56;
	t data_t[3] = 0x78;
	t data_t[4] = 0x9a;
	t data_t[5] = 0xbc;
	t data_t[6] = 0xde;
	t data_t[7] = 0xf0;
}
```

5.18.3 R_PG_LIN_Receive_LIN<*channel number*>

Definition

bool R_PG_LIN_Receive_LIN<*channel number*>
(uint8_t id, uint8_t data_count, bool checksum_enhanced)
<*channel number*>: 0

Description

Receive data

Parameter

uint8_t id	ID to be sent in the header
uint8_t data_count	The number of the data to be received
bool checksum_enhanced	Check sum select (0: Classic 1: Enhanced)

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_LIN_LIN<*channel number*>.c
<*channel number*>: 0

RPDL function

R_LIN_Transfer

Details

- Transmit the header and receive the response.
- Call R_PG_LIN_Set_LIN<*channel number*> before calling this function to set up the LIN module.
- This function needs to be called in LIN operation mode. To enter LIN operation mode from other modes, use R_PG_LIN_EnterOperationMode_LIN<*channel number*>.
- The status of transmission of the header and reception of the response can be acquired by R_PG_LIN_GetStatus_LIN<*channel number*>.
- The received data can be read by R_PG_LIN_ReadData_LIN<*channel number*>.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

// Storage area for the received data
uint8_t data_ r[8];

// Storage area for the frame/wake-up receive completion flag
bool frame_wakeup_ reception;

// Storage area for the error flag
bool error;

// Storage area for the check sum
bool check_sum;

void func(void)
{
      //Transmit the header and receive the response
      // (ID:3    Number of data:8    Check sum:Classic)
      R_PG_LIN_Receive_LIN0( 3, 8, 0 );
}

void Lin0IntFunc(void)
{
      //Get the frame/wake-up receive completion flag and error flag
      R_PG_LIN_GetStatus_LIN0(
            0,
            & frame_wakeup_reception,
            & error,
            0,
            0,
      );

      if( error ){
            //Error is detected
      }
      else if( frame_wakeup_reception ){
            // Frame/wake-up reception is completed

            //Read received data
            R_PG_LIN_ReadData_LIN0( data_r, 8 );

            //Read check sum
            R_PG_LIN_GetCheckSum_LIN0( & check_sum );
      }
}
```

5.18.4 R_PG_LIN_ReadData_LIN<*channel number*>

Definition

bool R_PG_LIN_ReadData_LIN<*channel number*>
(uint8_t * receive_data, uint8_t data_count)
<*channel number*>: 0

Description

Read data

Parameter

uint8_t * receive_data	The start address of the storage area for the expected data.
uint8_t data_count	The number of the data to be read

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output

R_PG_LIN_LIN<*channel number*>.c
<*channel number*>: 0

RPDL function

R_LIN_Read

Details

- Reads the data buffer registers.
- The response corresponding to the header sent by R_PG_LIN_Receive_LIN can be acquired by this function.
- The storage area specified by receive_data have to be big enough for the requested number of data.

Example

Refer to the example of R_PG_LIN_Receive_LIN<*channel number*>

5.18.5 R_PG_LIN_EnterResetMode_LIN<*channel number*>

Definition bool R_PG_LIN_EnterResetMode_LIN<*channel number*> (void)
<*channel number*>: 0

Description Enter LIN reset mode

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_LIN_LIN<*channel number*>.c
<*channel number*>: 0

RPDL function R_LIN_Control

Details
- Enters the LIN module in LIN reset mode
- To enter the LIN module in LIN wake-up mode or LIN operation mode from LIN self-test mode, enter LIN reset mode once by this function.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
    //Initialize LIN module    (LIN module is set to LIN operation mode)
    R_PG_LIN_Set_LIN0();

    //Enter LIN reset mode
    R_PG_LIN_EnterResetMode_LIN0();
}
```

5.18.6 R_PG_LIN_EnterOperationMode_LIN<*channel number*>

Definition: bool R_PG_LIN_EnterOperationMode_LIN<*channel number*> (void)
<*channel number*>: 0

Description: Enter LIN operation mode

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_LIN_LIN<*channel number*>.c
<*channel number*>: 0

RPDL function: R_LIN_Control

Details:
- Enters the LIN module in LIN operation mode
- To enter the LIN module in LIN operation mode from LIN self-test mode, enter LIN reset mode once by R_PG_LIN_EnterResetMode_LIN<*channel number*>.

Example:

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
    //Enter LIN reset mode
    R_PG_LIN_EnterResetMode_LIN0();

    //LIN operation mode
    R_PG_LIN_EnterOperationMode_LIN0();
}
```

5.18.7 R_PG_LIN_EnterWakeUpMode_LIN<*channel number*>

Definition bool R_PG_LIN_EnterWakeUpMode_LIN<*channel number*> (void)
<*channel number*>: 0

Description Enter LIN wake-up mode

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_LIN_LIN<*channel number*>.c
<*channel number*>: 0

RPDL function R_LIN_Control

Details
- Enters the LIN module in LIN wake-up mode
- To enter the LIN module in LIN wake-up mode from LIN self-test mode, enter LIN reset mode once by R_PG_LIN_EnterResetMode_LIN<*channel number*>.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
    // Enter LIN wake-up mode
    R_PG_LIN_EnterWakeUpMode_LIN0();

    //Wake-up transmission
    R_PG_LIN_WakeUpTransmit_LIN0();
}
```

5.18.8 R_PG_LIN_WakeUpTransmit_LIN<*channel number*>

Definition bool R_PG_LIN_WakeUpTransmit_LIN<*channel number*> (void)
<*channel number*>: 0

Description Transmit wake-up signals

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_LIN_LIN<*channel number*>.c
<*channel number*>: 0

RPDL function R_LIN_Control

Details
- Transmits wake-up signals
- This function needs to be called in LIN wake-up mode.
- The status of transmission of the wake-up signal can be acquired by R_PG_LIN_GetStatus_LIN<*channel number*>.

RPDL function Refer to the example of R_PG_LIN_EnterWakeUpMode_LIN<*channel number*>.

5.18.9 R_PG_LIN_WakeUpReceive_LIN<*channel number*>

Definition

bool R_PG_LIN_WakeUpReceive_LIN<*channel number*> (void)

<*channel number*>: 0

Description

Receive wake-up signals

Parameter

None

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_LIN_LIN<*channel number*>.c

<*channel number*>: 0

RPDL function

R_LIN_Control

Details

- Receives wake-up signals
- This function needs to be called in LIN wake-up mode.
- The status of reception of the wake-up signal can be acquired by R_PG_LIN_GetStatus_LIN<*channel number*>.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

// Storage area for the frame/wake-up receive completion flag
bool frame_wakeup_reception;

void func(void)
{
      //Initialize LIN module    (LIN module is set to LIN operation mode)
      R_PG_LIN_Set_LIN0();

      // Enter LIN wake-up mode
      R_PG_LIN_EnterWakeUpMode_LIN0();

      //Wake-up reception
      R_PG_LIN_WakeUpReceive_LIN0();

      //Wait for wake-up signal to be received
      do{
            R_PG_LIN_GetStatus_LIN0(
                  0,
                  & frame_wakeup_reception,
                  0,
                  0,
                  0,
            );
      }while( ! frame_wakeup_reception );
}
```

5.18.10 R_PG_LIN_GetCheckSum_LIN<*channel number*>

Definition: bool R_PG_LIN_GetCheckSum_LIN<*channel number*> (uint8_t * check_sum)
<*channel number*>: 0

Description: Get check sum

Parameter

uint8_t * check_sum	The start address of the storage area for check sum

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output: R_PG_LIN_LIN<*channel number*>.c
<*channel number*>: 0

RPDL function: R_LIN_GetStatus

Details
- Reads the check sum buffer register.

Example: Refer to the example of R_PG_LIN_Receive_LIN<*channel number*>

5.18.11 R_PG_LIN_EnterSelfTestMode_LIN<*channel number*>

Definition: bool R_PG_LIN_EnterSelfTestMode_LIN<*channel number*> (void)
<*channel number*>: 0

Description: Enter LIN self-test mode

Conditions for output: Use of LIN self test mode is enabled

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_LIN_LIN<*channel number*>.c
<*channel number*>: 0

RPDL function: R_LIN_Control

Details:

- Enters the LIN module in LIN self-test mode
- To enter LIN self-test mode from LIN wake-up mode or LIN operation mode, the LIN module enters LIN reset mode once in this function.

Example A case where the setting is made as follows.

- Lin0IntFunc was specified as the name of the LIN interrupt notification function.

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

// Storage area for the transmit data
uint8_t data_t[8];

// Storage area for the received data
uint8_t data_ r[8];

// Storage area for the frame/wake-up transmit completion flag
bool frame_wakeup_transmission;

// Storage area for the error flag
bool error;

void func(void)
{
    //Initialize LIN module    (LIN module is set to LIN operation mode)
    R_PG_LIN_Set_LIN0();

    //Enter LIN self-test mode
    R_PG_LIN_EnterSelfTestMode_LIN0();

    //Initialize transmit data
    InitData();

    //Start transmission test
    R_PG_LIN_Transmit_LIN0( 3, data_t, 8, 0 );
}

void Lin0IntFunc(void)
{
    //Get the frame/wake-up transmit completion flag and error flag
    R_PG_LIN_GetStatus_LIN0(
        & frame_wakeup_transmission,
        0,
        & error,
        0,
        0,
    );

    if( error ){
        //Error is detected
    }
    else if( frame_wakeup_transmission ){
        //Transmission completion

        //Read the data buffer registers
        R_PG_LIN_ReadData_LIN0( data_r, 8 );
    }
}

void InitData(void)
{
    t data_t[0] = 0x12;
    t data_t[1] = 0x34;
    t data_t[2] = 0x56;
    t data_t[3] = 0x78;
    t data_t[4] = 0x9a;
    t data_t[5] = 0xbc;
    t data_t[6] = 0xde;
    t data_t[7] = 0xf0;
}
```

5.18.12 R_PG_LIN_WriteCheckSum_LIN<*channel number*>

Definition: bool R_PG_LIN_WriteCheckSum_LIN<*channel number*> (uint8_t check_sum)
<*channel number*>: 0

Description: Write check sum

Conditions for output: Use of LIN self test mode is enabled

Parameter

uint8_t check_sum	Check sum value to be written to check sum buffer register

Return value

true	Setting was made correctly
false	Setting failed

File for output: R_PG_LIN_LIN<*channel number*>.c
<*channel number*>: 0

RPDL function: R_LIN_Control

Details
- Sets check sum buffer register.
- Use this function to set check sum in reception of LIN self-test mode.

Example A case where the setting is made as follows.

- Lin0IntFunc was specified as the name of the LIN interrupt notification function.

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

// Storage area for the received data
uint8_t data_ r[8];

// Storage area for the frame/wake-up receive completion flag
bool frame_wakeup_ reception;

// Storage area for the error flag
bool error;

void func(void)
{
	//Initialize LIN module	(LIN module is set to LIN operation mode)
	R_PG_LIN_Set_LIN0();

	//Enter LIN self-test mode
	R_PG_LIN_EnterSelfTestMode_LIN0();

	//Set data buffer registers
	SetData();

	// Start reception test
	R_PG_LIN_Receive_LIN0( 3, 8, 0 );
}

void Lin0IntFunc(void)
{
	//Get the frame/wake-up transmit completion flag and error flag
	R_PG_LIN_GetStatus_LIN0(
		0,
		& frame_wakeup_reception,
		& error,
		0,
		0,
	);

	if( error ){
		// Error is detected
	}
	else if( frame_wakeup_reception ){
		//Reception completion

		//Read the data buffer registers
		R_PG_LIN_ReadData_LIN0( data_r, 8 );
	}
}

void SetData(void)
{
	// Set data buffer registers
	*((uint8_t*)(0x94018))=data_t[0]; //LDB1
	*((uint8_t*)(0x94019))=data_t[1]; //LDB2
	*((uint8_t*)(0x9401A))=data_t[2]; //LDB3
	*((uint8_t*)(0x9401B))=data_t[3]; //LDB4
	*((uint8_t*)(0x9401C))=data_t[4]; //LDB5
	*((uint8_t*)(0x9401D))=data_t[5]; //LDB6
	*((uint8_t*)(0x9401E))=data_t[6]; //LDB7
	*((uint8_t*)(0x9401F))=data_t[7]; //LDB8

	//Set check sum
	R_PG_LIN_WriteCheckSum_LIN0 ( 0xc3 )
}
```

5.18.13 R_PG_LIN_GetMode_LIN<*channel number*>

Definition

bool R_PG_LIN_GetMode_LIN<*channel number*> (uint8_t * mode)

<*channel number*>: 0

Description

Get current operation mode

Parameter

uint8_t * mode	The address of the storage area for value indicates the operation mode Relationship between operation mode and value

LIN reset mode	0x00
LIN wake-up mode	0x01
LIN operation mode	0x03
LIN self-test mode	0x04

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output

R_PG_LIN_LIN<*channel number*>.c

<*channel number*>: 0

RPDL function

R_LIN_GetStatus

Details

- Acquires the current operation mode of LIN module.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

uint8_t mode    // Storage area for value indicates the operation mode

void func(void)
{
      R_PG_LIN_GetMode_LIN0( &mode ) //Get the current operation mode

      switch( mode ){
            case 0x00:
                  //LIN reset mode
                  break;
            case 0x01:
                  //LIN wake-up mode
                  break;
            case 0x03:
                  //LIN operation mode
                  break;
            case 0x04:
                  //LIN self-test mode
                  break;
            default;
                  break;
      }
}
```

5.18.14 R_PG_LIN_GetStatus_LIN<*channel number*>

Definition

bool R_PG_LIN_GetStatus_LIN<*channel number*>
(bool * frame_wakeup_transmission, bool * frame_wakeup_reception, bool * error,
bool * data1_reception, bool * header_transmission)
<*channel number*>: 0

Description

Get the status of LIN module

Parameter

bool * frame_wakeup_transmission	The address of the storage area for frame/wake-up transmit completion flag
bool * frame_wakeup_reception	The address of the storage area for frame/wake-up receive completion flag
bool * error	The address of the storage area for error detection flag
bool * data1_reception	The address of the storage area for data 1 receive completion flag
bool * header_transmission	The address of the storage area for header transmit completion flag

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output

R_PG_LIN_LIN<*channel number*>.c
<*channel number*>: 0

RPDL function

R_LIN_GetStatus

Details

- Acquires the status of LIN module.
- Specify the address of storage area for the flags to be acquired. Specify 0 for a flag that is not required.

Example

Refer to the example of R_PG_LIN_Transmit_LIN0, R_PG_LIN_Receive_LIN0, R_PG_LIN_WakeUpReceive_LIN0, R_PG_LIN_EnterSelfTestMode_LIN0, and R_PG_LIN_WriteCheckSum_LIN0.

5.18.15 R_PG_LIN_GetErrorStatus_LIN<*channel number*>

Definition

bool R_PG_LIN_GetErrorStatus_LIN<*channel number*>
(bool * bit_error, bool * bus_error, bool * frame_timeout,
bool * framing, bool * check_sum_error)
<*channel number*>: 0

Description

Get the status of error detection

Parameter

bool * bit_error	The address of the storage area for bit error flag
bool * bus_error	The address of the storage area for physical bus error flag
bool * frame_timeout	The address of the storage area for frame timeout error flag
bool * framing	The address of the storage area for framing error flag
bool * check_sum_error	The address of the storage area for check sum error flag

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output

R_PG_LIN_LIN<*channel number*>.c
<*channel number*>: 0

RPDL function

R_LIN_GetStatus

Details

- Acquires the status of error detection
- Specify the address of storage area for the flags to be acquired. Specify 0 for a flag that is not required.

Example

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

bool error;              // Storage area for error flag

bool bit_error;          // Storage area for bit error flag
bool bus_error;          // Storage area for physical bus error flag
bool frame_timeout;      // Storage area for frame timeout error flag
bool framing;            // Storage area for framing error flag
bool check_sum_error; // Storage area for check sum error flag

void func(void)
{
    // Get the error flag
    R_PG_LIN_GetStatus_LIN0(
        0,
        0,
        & error,
        0,
        0,
    );

    if( error ){
        //Error is detected

        //Get the status of error detection
        R_PG_LIN_GetErrorStatus_LIN0(
            & bit_error,
            & bus_error,
            & frame_timeout,
            & framing,
            & check_sum_error       );

        if( bit_error ){
            // Bit error is detected
        }
        if( bus_error ){
            // Physical bus error is detected
        }
        if( frame_timeout ){
            // Frame timeout error is detected
        }
        if( framing ){
            // Framing error is detected
        }
        if( check_sum_error ){
            // Check sum error is detected
        }
    }
}
```

5.18.16 R_PG_LIN_StopModule_LIN<*channel number*>

Definition: bool R_PG_LIN_StopModule_LIN<*channel number*> (void)
<*channel number*>: 0

Description: Shut down a LIN module

Parameter: None

Return value:

true	Shutting down succeeded
false	Shutting down failed

File for output: R_PG_LIN_LIN<*channel number*>.c
<*channel number*>: 0

RPDL function: R_LIN_Destroy

Details:

- Stops a LIN module and places it in the module-stop state.

Example:

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
      // Shut down a LIN module
      R_PG_LIN_StopModule_LIN0();
}
```

5.19 12-Bit A/D Converter (S12ADA)

5.19.1 R_PG_ADC_12_Set_S12ADA<*unit number*>

Definition: bool R_PG_ADC_12_Set_S12ADA<*unit number*> (void) <*unit number*>: 0 to 1

Description: Set up the 12-Bit A/D Converter

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_ADC_12_S12ADA<*unit number*>.c <*unit number*>: 0 to 1

RPDL function: R_ADC_12_CreateUnit, R_ADC_12_Set

Details:

- Releases the 12-Bit A/D converter from the module-stop state, makes initial settings, and places it in the conversion-start trigger-input wait state. When the software trigger is selected to start conversion, conversion is started by calling R_PG_ADC_12_StartConversionSW_S12ADA<*unit number*>.
- Function R_PG_Clock_Set must be called before calling this function.
- The input direction is set for pins used as analog inputs and the input buffers for the pins are disabled.
- The A/D-conversion end interrupt is set in this function. When the name of the interrupt notification function has been specified in the GUI, if an interrupt occurs in the CPU, the function having the specified name will be called. Create the interrupt notification function as follows:
  void <*name of the interrupt notification function*> (void)
  For the interrupt notification function, note the contents of 5.21, Notes on Notification Functions.

Example:

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
    R_PG_Clock_Set();                //Set up the clocks
    R_PG_ADC_12_Set_S12ADA0();    //Set up the 12-Bit A/D converter
}
```

5.19.2 R_PG_ADC_12_Set

Definition bool R_PG_ADC_12_Set(void)

Description Set the programmable gain amplifier

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_ADC_12.c

RPDL function R_ADC_12_CreateChannel

Details
- The gain is set in this function.

Example A case where the setting is made as follows.
- The setting that uses the programmable gain amplifier has been specified in the GUI.

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
    R_PG_Clock_Set();                 //Set up the clocks
    R_PG_ADC_12_Set_S12ADA0();        //Set up the 12-Bit A/D converter

    // Set the programmable gain amplifier
    R_PG_ADC_12_Set();

    // Start A/D conversion by the software trigger
    R_PG_ADC_12_StartConversionSW_S12ADA0();
}
```

5.19.3 R_PG_ADC_12_StartConversionSW_S12ADA<*unit number*>

Definition: bool R_PG_ADC_12_StartConversionSW_S12ADA<*unit number*> (void)
<*unit number*>: 0 to 1

Description: Start A/D conversion (Software trigger)

Conditions for output: Setting of the A/D converter and specification of the software trigger as the activation source

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_ADC_12_S12ADA<*unit number*>.c <*unit number*>: 0 to 1

RPDL function: R_ADC_12_Control

Details:
- Starts A/D conversion by an A/D converter for which the software trigger is selected as the activation source.

Example: A case where the setting is made as follows.
- The software trigger is selected as the conversion start trigger.

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
    R_PG_Clock_Set();                   //Set up the clocks
    R_PG_ADC_12_Set_S12ADA0();          //Set up the 12-Bit A/D converter

    // Start A/D conversion by the software trigger
    R_PG_ADC_12_StartConversionSW_S12ADA0();
}
```

5.19.4 R_PG_ADC_12_StopConversion_S12ADA<*unit number*>

Definition bool R_PG_ADC_12_StopConversion_S12ADA<*unit number*> (void)
<*unit number*>: 0 to 1

Description Stop A/D conversion

Parameter None

Return value

true	Stopping conversion succeeded.
false	Stopping conversion failed.

File for output R_PG_ADC_12_S12ADA<*unit number*>.c <*unit number*>: 0 to 1

RPDL function R_ADC_12_Control

Details

- A/D conversion in the continuous scan mode can be stopped. Except the continuous scan mode, this function need not be called after A/D conversion has ended.
  After this function has stopped A/D conversion, continuous scanning is resumed on input of the A/D-conversion start trigger. To end continuous scanning, stop the A/D conversion unit by calling R_PG_ADC_12_StopModule_S12ADA<*unit number*>.

Example A case where the setting is made as follows.

- The continuous scan mode is selected as the operation mode.

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

uint16_t data;    //Storage for the result of A/D conversion

void func1(void)
{
      R_PG_Clock_Set();                 //Set up the clocks
      R_PG_ADC_12_Set_S12ADA0();        //Set up the 12-Bit A/D converter
}

void func2(void)
{
      //Stop the continuous scan
      R_PG_ADC_12_StopConversion_S12ADA0();

      // Acquire the result of A/D conversion
      R_PG_ADC_12_GetResult_S12ADA0(&data);

      //Stop the 12-Bit A/D Converter
      R_PG_ADC_12_StopModule_S12ADA0();
}
```

5.19.5 R_PG_ADC_12_GetResult_S12ADA<*unit number*>

Definition bool R_PG_ADC_12_GetResult_S12ADA<*unit number*> (uint16_t * result)
<*unit number*>: 0 to 1

Description Acquire the result of A/D conversion

Parameter

uint16_t * result	The address of the storage area for the result of A/D conversion

Return value

true	Acquisition of the result succeeded
false	Acquisition of the result failed

File for output R_PG_ADC_12_S12ADA<*unit number*>.c <*unit number*>: 0 to 1

RPDL function R_ADC_12_Read

Details

- From 1 to 5 two-byte memory slots are required. The number depends on the “Input channel selection” and A/D-conversion start trigger.
- If “Double Trigger” has been specified for channel ANn00 (n = 0 or 1) in the GUI, the first memory slot contains the result from channel ANn00 (n = 0 or 1) triggered by “A Trigger” and the last one contains the result from channel ANn00 (n = 0 or 1) triggered by “B Trigger”.

  Double Trigger : (A Trigger / B Trigger)
  No.1: (TRG4AN / TRG4BN)
  No.2: (TRG7AN / TRG7BN)
  No.3: (GTADTRA0N / GTADTRB0N)
  No.4: (GTADTRA1N / GTADTRB1N)
  No.5: (GTADTRA2N / GTADTRB2N)
  No.6: (GTADTRA3N / GTADTRB3N)

  Example of data format:
  Two channel scan mode.
  AN000 selected as group 0, triggered by “Double Trigger”;
  AN001, AN002 and AN003 selected as group 1, triggered by TRGA2N.

  result[0] : The conversion result from AN000 triggered by “A Trigger” ADDR0A
  result[1] : The conversion result from AN001 ADDR1
  result[2] : The conversion result from AN002 ADDR2
  result[3] : The conversion result from AN003 ADDR3
  result[4] : The conversion result from AN000 triggered by “B Trigger” ADDR0B

- When the MTU3 or GPT trigger is selected, the last memory slot contains the value of ADDR0B though ADDR0B is not used except for “Double Trigger”.
- When A/D conversion is in progress at the time of calling this function and a name for the interrupt notification function has not been specified through the GUI, the function waits until the end of A/D conversion before reading the result.

Example A case where the setting is made as follows.

- Two channel scan mode.
  AN000 selected as group 0, triggered by (TRG4AN / TRG4BN);
  AN001, AN002 and AN003 selected as group 1, triggered by TRG7BN.

```
//Include “R_PG_<project name>.h” to use this function.
#include "R_PG_default.h"

uint16_t result[5];          //Storage for the result of A/D conversion

void func(void)
{
      R_PG_Clock_Set();                 //Set up the clocks
      R_PG_ADC_12_Set_S12ADA0();        //Set up the 12-Bit A/D converter
}

//The A/D conversion end interrupt notification function
void S12ad0IntFunc(void)
{
      // Acquire the result of A/D conversion
      R_PG_ADC_12_GetResult_S12ADA0( result );
}
```

5.19.6 R_PG_ADC_12_GetResult_SelfDiag_S12AD<*unit number*>

Definition bool R_PG_ADC_12_GetResult_SelfDiag_S12AD<*unit number*> (uint16_t * result)
<*unit number*>: 0 to 1

Description Gets the result of A/D conversion as part of self diagnosis by the A/D converter

Parameter

uint16_t * result	Destination for storage of the result of A/D conversion

Return value

true	Acquisition of the result succeeded.
false	Acquisition of the result failed.

File for output R_PG_ADC_12_S12ADA<*unit number*>.c <*unit number*>: 0 to 1

RPDL function R_ADC_12_Read

Details

- Acquires the result of A/D conversion performed as part of self diagnosis.
- When you use the self-diagnosis facility, self diagnosis takes place once at the beginning of each round of scanning with A/D conversion of one of the three voltages generated within the A/D converter.
- The acquired result of A/D conversion includes self-diagnosis status information*, which is in either of the following formats.

When the data placement selected through the GUI is right-alignment
b15-b14: Self-diagnosis status information*
b11-b0: Result of A/D conversion as part of self diagnosis
When the data placement selected through the GUI is left-alignment
b15-b4: Result of A/D conversion as part of self diagnosis
b1-b0: Self-diagnosis status information*

Note: The self-diagnosis status information has the following meanings.
b'00: Self diagnosis has not been performed.
b'01: Self diagnosis on 0[V] voltage has been performed.
b'10: Self diagnosis on VREFH0 × 1/2 voltage has been performed.
b'11: Self diagnosis on VREFH0 voltage has been performed.

Example The following settings have been made through the GUI.

- Select the single scan mode.
- Select AN000 and AN003 as analog input pins.
- Select the software trigger as the activation source.
- Select right-alignment for data placement.
- Enable the self-diagnosis facility.
- Specify S12ad0AIntFunc as the A/D-conversion end interrupt notification function.

```
// Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

uint16_t result_selfdiag;   // Destination for storing the result of A/D conversion as part of
                              // self diagnosis
uint16_t adrd_ad;           // Destination for storing the result of 12-bit A/D conversion
uint16_t adrd_diagst;       // Destination for storing the self-diagnosis status information
uint16_t result[5];         // Destination for storing the result of A/D conversion on AN000
                              // and AN003
uint16_t result_an000;      // Destination for storing the result of A/D conversion on AN000
uint16_t result_an003;      // Destination for storing the result of A/D conversion on AN003

void func(void)
{
    R_PG_Clock_Set();               // The clock-generation circuit has to be set first.
    R_PG_ADC_12_Set_S12AD0();   // Set up the 12-bit A/D converter (S12AD0).

    // A software trigger starts A/D conversion.
    R_PG_ADC_12_StartConversionSW_S12AD0();
}

// A/D-conversion end interrupt notification function
void S12ad0AIntFunc(void)
{
    // Acquire the results of A/D conversion as part of self diagnosis.
    R_PG_ADC_12_GetResult_SelfDiag_S12AD0( &result_selfdiag );

    adrd_ad = (result_selfdiag & 0x0fff);
    adrd_diagst = (result_selfdiag >> 14);

    // Acquire the result of A/D conversion on AN000 and AN003.
    R_PG_ADC_12_GetResult_S12AD0( result );

    result_an000 = result[0];
    result_an003 = result[3];
}
```

5.19.7 R_PG_ADC_12_StopModule_S12ADA<*unit number*>

Definition bool R_PG_ADC_12_StopModule_S12ADA<*unit number*> (void)
<*unit number*>: 0 to 1

Description Shut down the 12-Bit A/D converter

Parameter None

Return value

true	Shutting down succeeded
false	Shutting down failed

File for output R_PG_ADC_12_S12ADA<*unit number*>.c <*unit number*>: 0 to 1

RPDL function R_ADC_12_Destroy

Details
- Stops the 12-Bit A/D converter and places it in the module-stop state. (Power consumption decrease functoin)
- When two units have been used, even if this function is called for one unit, it doesn't shift in the state of the module-stop state. Afterwards, both units shift in the state of the module-stop state when this function is called for the other unit.

Example Refer to the example of R_PG_ADC_12_StopConversion_S12ADA<*unit number*>

5.20 10-Bit A/D Converter (ADA)

5.20.1 R_PG_ADC_10_Set_AD<*unit number*>

Definition: bool R_PG_ADC_10_Set_AD<*unit number*> (void) <*unit number*>: 0

Description: Set up the 10-Bit A/D Converter (ADA)

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_ADC_10_AD<*unit number*>.c <*unit number*>: 0

RPDL function: R_ADC_10_Create

Details:

- Releases an A/D converter from the module-stop state, makes initial settings, and places it in the conversion-start trigger-input wait state.
- Call R_PG_ADC_10_StartConversionSW_AD<*unit number*> to start the A/D-conversion by the software trigger.
  Function R_PG_Clock_Set must be called before calling this function.
- The input direction is set for pins used as analog inputs and the input buffers for the pins are disabled.
- The A/D-conversion end interrupt is set in this function. When the name of the interrupt notification function has been specified in the GUI, if an interrupt request is conveyed to the CPU, the function having the specified name will be called. Create the interrupt notification function as follows:
  void <name of the interrupt notification function> (void)
  For the interrupt notification function, note the contents of 5.21, Notes on Notification Functions.

Example: The hardware trigger has been specified in the GUI.
Ad0IntFunc has been specified as the name of the A/D-conversion end interrupt notification function in the GUI.

```
#include "R_PG_default.h" //Include "R_PG_<project name>.h" to use this function.
uint16_t data;    //Destination for storage of the result of A/D conversion

void func(void)
{
      R_PG_Clock_Set();              //The clock-generation circuit has to be set first.
      R_PG_ADC_10_Set_AD0();         //Set up ADA.
}

//AD-conversion end interrupt notification function
void Ad0IntFunc(void)
{
      R_PG_ADC_10_GetResult_AD0(&data)       //Acquire the result of A/D conversion.
}
```

5.20.2 R_PG_ADC_10_StartConversionSW_AD<*unit number*>

Definition bool R_PG_ADC_10_StartConversionSW_AD<*unit number*> (void)
<*unit number*>: 0

Description Start the A/D conversion (Software trigger)

Parameter None

Return value

true	Triggering the conversion succeeded.
false	Triggering the conversion failed.

File for output R_PG_ADC_10_AD<*unit number*>.c
<*unit number*>: 0

RPDL function R_ADC_10_Control

Details
- Call this function when you use the software trigger.
- Even when the hardware trigger has been selected, the A/D-conversion can be started by calling this function.

Example The software trigger has been specified in the GUI.

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

void func(void)
{
    R_PG_Clock_Set();              //The clock-generation circuit has to be set first.
    R_PG_ADC_10_Set_AD0();         //Set up AD0.

    //Start A/D conversion by the software trigger
    R_PG_ADC_10_StartConversionSW_AD0();
}
```

5.20.3 R_PG_ADC_10_StopConversion_AD<*unit number*>

Definition bool R_PG_ADC_10_StopConversion_AD<*unit number*> (void)
<*unit number*>: 0

Description Stop the A/D conversion

Parameter None

Return value

true	Stopping the conversion succeeded.
false	Stopping the conversion failed.

File for output R_PG_ADC_10_AD<*unit number*>.c
<*unit number*>: 0

RPDL function R_ADC_10_Control

Details

- A/D conversion can be stopped in the continuous scan mode. In the single mode and single-cycle scan mode, this function need not be called after A/D conversion has ended. After this function has stopped A/D conversion, continuous scanning is resumed on input of the A/D-conversion start trigger. To end continuous scanning, stop the A/D conversion unit by calling R_PG_ADC_10_StopModule_AD<*unit number*>.

Example The continuous scan mode has been specified in the GUI.

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"

uint16_t data;    //Destination for storage of the result of A/D conversion

void func1(void)
{
      R_PG_Clock_Set();                 //The clock-generation circuit has to be set first.
      R_PG_ADC_10_Set_AD0();        //Set up AD0.
}

void func2(void)
{
      //Stop continuous scanning.
      R_PG_ADC_10_StopConversion_AD0();

      //Acquire the result of A/D conversion.
      R_PG_ADC_10_GetResult_AD0(&data);

      //Stop the A/D converter.
      R_PG_ADC_10_StopModule_AD0();
}
```

5.20.4 R_PG_ADC_10_GetResult_AD<*unit number*>

Definition bool R_PG_ADC_10_GetResult_AD<*unit number*> (uint16_t * result)

<*unit number*>: 0

Description Get the result of A/D conversion

Parameter

uint16_t * result	Destination for storage of the result of A/D conversion

Return value

true	Acquisition of the result succeeded.
false	Acquisition of the result failed.

File for output R_PG_ADC_10_AD<*unit number*>.c <*unit number*>: 0

RPDL function R_ADC_10_Read

Details

- The amount of data to be acquired depends on the number of A/D-conversion channels that are in use. Reserve the area required for storing the result of A/D conversion for the given number of channels.
- When A/D conversion is in progress at the time of calling this function and a name for the interrupt notification function has not been specified through the GUI, the function waits until the end of A/D conversion before reading the result.

Example Four channels (AN0 to AN3) are in use.
Ad0IntFunc has been specified as the name of the A/D-conversion end interrupt notification function in the GUI.

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"
void func(void)
{
    R_PG_Clock_Set();              //The clock-generation circuit has to be set first.
    R_PG_ADC_10_Set_AD0();         //Set up AD0.
}
//AD-conversion end interrupt notification function
void Ad0IntFunc(void)
{
    uint16_t data[4];    //Result of A/D conversion on all channels
    uint16_t data_an0;   //Result of A/D conversion on AN0
    uint16_t data_an1;   //Result of A/D conversion on AN1
    uint16_t data_an2;   //Result of A/D conversion on AN2
    uint16_t data_an3;   //Result of A/D conversion on AN3
    R_PG_ADC_10_GetResult_AD0(data);     //Acquire the results of A/D conversion.
    data_an0 = data[0];
    data_an1 = data[1];
    data_an2 = data[2];
    data_an3 = data[3];
}
```

5.20.5 R_PG_ADC_10_SetSelfDiag_VREF_<*voltage*>_AD<*unit number*>

Definition: bool R_PG_ADC_10_SetSelfDiag_VREF_<*voltage*>_AD<*unit number*> (void)
<*voltage*>: 0, 0_5, 1 (0:Vref*0, 0_5:Vref/2, 1:Vref) <*unit number*>: 0

Description: Set up the A/D self-diagnostic function

Conditions for output: The self-diagnostic function is enabled

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_ADC_10_AD<*unit number*>.c <*unit number*>: 0

RPDL function: R_ADC_10_Create

Details:

- Setsup the A/D self-diagnostic function.
- In this function, the A/D conversion mode is set to the single mode and the conversion start trigger is set to the software trigger.
- To re-set the A/D converter, call R_PG_ADC_10_Set_AD<*unit number*>.
- To start the self-diagnostic, call R_PG_ADC_10_StartConversionSW_AD<*unit number*> and to get the result of self-diagnostic, call R_PG_ADC_10_GetResult_AD<*unit number*>.

Example:

```
//Include "R_PG_<project name>.h" to use this function.
#include "R_PG_default.h"
uint16_t SelfDiagnostic_0()
{
      uint16_t result;
      R_PG_ADC_10_SetSelfDiag_VREF_0_AD0();
      R_PG_ADC_10_StartConversionSW_AD0();
      R_PG_ADC_10_GetResult_AD0 (&result);
      return result;
}
uint16_t SelfDiagnostic_0_5()
{
      uint16_t result;
      R_PG_ADC_10_SetSelfDiag_VREF_0_5_AD0();
      R_PG_ADC_10_StartConversionSW_AD0();
      R_PG_ADC_10_GetResult_AD0 (&result);
      return result;
}
uint16_t SelfDiagnostic_1()
{
      uint16_t result;
      R_PG_ADC_10_SetSelfDiag_VREF_1_AD0();
      R_PG_ADC_10_StartConversionSW_AD0();
      R_PG_ADC_10_GetResult_AD0 (&result);
      return result;
}
```

5.20.6 R_PG_ADC_10_StopModule_AD<*unit number*>

Definition: bool R_PG_ADC_10_StopModule_AD<*unit number*> (void)
<*unit number*>: 0

Description: Shut down the 10-Bit A/D Converter (ADA)

Parameter: None

Return value:

true	Shutting down succeeded
false	Shutting down failed

File for output: R_PG_ADC_10_AD<*unit number*>.c
<*unit number*>: 0

RPDL function: R_ADC_10_Destroy

Details:
- Stops an A/D converter and places it in the module-stop state. (Power consumption decrease functoin)

Example: Refer to the example of R_PG_ADC_10_StopConversion_AD<*unit number*>

5.21 Notes on Notification Functions

5.21.1 Interrupts and processor mode

The RX CPU has two processor modes; supervisor and user.The driver functions will be executed by the CPU in user mode.However any notification functions which are called by the interrupt handlers in Renesas Peripheral Driver Library will be executed by the CPU in supervisor mode.This means that the privileged CPU instructions (RTFI, RTE and WAIT) can be executed by the notification function and any function that is called by the notification function.
The user must:

- Avoid using the RTFI and RTE instructions.

  These instructions are issued by the API interrupt handlers, so there should be no need for the user's code to use these instructions.
- Use the wait() intrinsic function with caution.

  This instruction is used by some API functions as part of power management, so there should be no need for the user's code to use this instruction.

More information on the processor modes can be found in §1.4 of the RX Family software manual.

5.21.2 Interrupts and DSP instructions

The accumulator (ACC) register is modified by the following instructions:

- DSP (MACHI, MACLO, MULHI, MULLO, MVTACHI, MVTACLO and RACW).
- Multiply and multiply-and-accumulate (EMUL, EMULU, FMUL, MUL, and RMPA)

The accumulator (ACC) register is not pushed onto the stack by the interrupt handlers in Renesas Peripheral Driver Library.
If DSP instructions are being utilised in the users' code, notification functions which are called by the interrupt handlers in Renesas Peripheral Driver Library should either

- Avoid using instructions which modify the ACC register.
- Take a copy of the ACC register and restore it before exiting the callback function.

6. Registering Files with the IDE and Building Them

Note the following points when registering the files generated by the Peripheral Driver Generator with the IDE(High-performance Embedded Workshop, CubeSuite+ or e2 studio) and building them.

(1) Source files generated by the Peripheral Driver Generator do not include a startup program. For this reason, you need to create a startup program by specifying [Application] as the project type during the process of creating a IDE project.

(2) Source files registered by the Peripheral Driver Generator with the IDE include an interrupt handler and vector table. Since the interrupt handler and vector table must not overlap with those included in the startup program created by using the IDE, intprg.c and vecttbl.c are excluded from the set of files that are included in the build.Interrupt_handler.c and vector_table.c are made the target in case of e2studio.

(3) Source files Interrupt_xxx.c, which includes the interrupt handler that the Peripheral Driver Generator registers with the IDE, is overwritten when the Peripheral Driver Generator generates source files.

(4) The Renesas Peripheral Driver Library is produced using the default compiler options (except that [Double precision] is selected for [Precision of double]). If you specify the compiler options other than the defaults in your project, you have to utilize Renesas Peripheral Driver Library source under your responsibility.

(5) The Renesas Peripheral Driver Library has been built specifying double-precision floating point. Therefore, to build the user program with Peripheral Driver Generator-generated files, specify double-precision floating point option in builder settings of IDE as follows. It's unnecessary at the time of e2 studio use.

CubeSuite+

1. Open the [CC-RX Property] by double-clicking [CC-RX(Build Tool)] in project tree.
2. In the [CPU] category, select [Handles in double precision] for [Precision of the double type and long double type].

High-performance Embedded Workshop

1. Select [Build]->[RX Standard Toolchain] from main menu to open the [RX Standard Toolchain] dialog box.
2. Select the [CPU] tab.
3. Click the [Details] button to open the [CPU details] dialog box.
4. Select [Double precision] for [Precision of double].

(6) The RPDL library use FIXEDVECT section that address is 0xFFFFFFD0. Therefore, to build the user program with PDG-generated files, specify the linker option in builder setting of IDE as follows. It’s necessary at the time of e2 studio use.

1. Select the project on Project Explorer.
2. Select [File]->[Properties] from main menu to open the [Properties] window.
3. Select [C/C++ build] ->[Settings]
4. Select [All configurations] for [Configuration]
5. Select [Linker] -> [Section] to show [Section viewer]
6. Set the address of the FIXEDVECT section as 0xFFFFFFD0.

Appendix 1. Pin Functions for which the Allocation Can be Changed

Table a-1.1 112-pin LQFP (the Upper Row of Each Pair is the Default Selection)

Peripheral module	Pin function	Selection of assignment	Pin No.
ICU (External Interrupts)	IRQ0	P10/MTCLKD/IRQ0	110
		PE5/IRQ0	1
		PG0/IRQ0/TRSYNC	59
	IRQ1	P11/MTCLKC/IRQ1	109
		PE4/MTCLKC/IRQ1/POE10#	8
		PG1/IRQ1/TRDATA0	58
	IRQ2	PE3/MTCLKD/IRQ2/POE11#	9
		PG2/IRQ2/TRDATA1	57
MTU3	MTCLKA	P33/MTIOC3A/MTCLKA/SSL3	67
	*1	P21/ADTRG1#/MTCLKA/IRQ6	76
	MTCLKB	P32/MTIOC3C/MTCLKB/SSL2	68
	*1	P20/ADTRG0#/MTCLKB/IRQ7	77
	MTCLKC	P31/MTIOC0A/MTCLKC/SSL1	70
		P11/MTCLKC/IRQ1	109
	*1	PE4/MTCLKC/IRQ1/POE10#	8
	MTCLKD	P30/MTIOC0B/MTCLKD/SSL0	72
		P10/MTCLKD/IRQ0	110
	*1	PE3/MTCLKD/IRQ2/POE11#	9
MTU3_0	MTIOC0A	PB3/MTIOC0A/SCK0	35
		P31/MTIOC0A/MTCLKC/SSL1	70
	MTIOC0B	PB2/MTIOC0B/TXD0/SDA	36
		P30/MTIOC0B/MTCLKD/SSL0	72
POE	POE10#	PE2/NMI/POE10#	15
		PE4/MTCLKC/IRQ1/POE10#	8
GPT0	GTIOC0A	P71/MTIOC3B/GTIOC0A	65
	*2	PD7/GTIOC0A/CTX/SSL1	18
	GTIOC0B	P74/MTIOC3D/GTIOC0B	62
	*2	PD6/GTIOC0B/SSL0	19
GPT1	GTIOC1A	P72/MTIOC4A/GTIOC1A	64
	*2	PD5/GTIOC1A/RXD1	20
	GTIOC1B	P75/MTIOC4C/GTIOC1B	61
	*2	PD4/GTIOC1B/SCK1	21
GPT2	GTIOC2A	P73/MTIOC4B/GTIOC2A	63
	*2	PD3/GTIOC2A/TXD1	22
	GTIOC2B	P76/MTIOC4D/GTIOC2B	60
	*2	PD2/GTIOC2B/MOSI	23
SCI2	TXD2	PB5/CTX/TXD2	31
	*3	P81/MTIC5V/TXD2	106
	RXD2	PB6/CRX/RXD2	30
	*3	P80/MTIC5W/RXD2	107
	SCK2	PB7/SCK2	29
	*3	P82/MTIC5U/SCK2	105
RSPI0	RSPCK	P24/RSPCK	73
		PA4/ADTRG0#/MTIOC1B/RSPCK	40
	*4	PD0/GTIOC3B/RSPCK	25
	MOSI	P23/CTX/LTX/MOSI	74
		PB0/MTIOC0D/MOSI	38

	*4	PD2/GTIOC2B/MOSI	23
	MISO	P22/ADTRG#/CRX/LRX/MISO	75
		PA5/ADTRG1#/MTIOC1A/MISO	39
	*4	PD1/GTIOC3A/MISO	24
	SSL0	P30/MTIOC0B/MTCLKD/SSL0	72
		PA3/MTIOC2A/SSL0	41
	*4	PD6/GTIOC0B/SSL0	19
	SSL1	P31/MTIOC0A/MTCLKC/SSL1	70
		PA2/MTIOC2B/SSL1	42
	*4	PD7/GTIOC0A/CTX/SSL1	18
	SSL2	P32/MTIOC3C/MTCLKB/SSL2	68
		PA1/MTIOC6A/SSL2	43
	*4	PE0/CRX/SSL2	17
	SSL3	P33/MTIOC3A/MTCLKA/SSL3	67
		PA0/MTIOC6C/SSL3	44
	*4	PE1/SSL3	16
S12ADA0	ADTRG0#	PA4/ADTRG0#/MTIOC1B/RSPCK	40
		P20/ADTRG0#/MTCLKB/IRQ7	77
S12ADA1	ADTRG1#	PA5/ADTRG1#/MTIOC1A/MISO	39
		P21/ADTRG1#/MTCLKA/IRQ6	76

*1 to 4 The settings are linked together

Table a-1.2 100-pin LQFP (the Upper Row of Each Pair is the Default Selection)

Peripheral module	Pin function	Selection of assignment	Pin No.
ICU (External Interrupts)	IRQ0	P10/MTCLKD/IRQ0	100
		PE5/IRQ0	1
	IRQ1	P11/MTCLKC/IRQ1	99
		PE4/MTCLKC/IRQ1/POE10#	8
MTU3	MTCLKA	P33/MTIOC3A/MTCLKA/SSL3	58
	*1	P21/ADTRG1#/MTCLKA/IRQ6	67
	MTCLKB	P32/MTIOC3C/MTCLKB/SSL2	59
	*1	P20/ADTRG0#/MTCLKB/IRQ7	68
	MTCLKC	P31/MTIOC0A/MTCLKC/SSL1	61
		P11/MTCLKC/IRQ1	99
	*1	PE4/MTCLKC/IRQ1/POE10#	8
	MTCLKD	P30/MTIOC0B/MTCLKD/SSL0	63
		P10/MTCLKD/IRQ0	100
	*1	PE3/MTCLKD/IRQ2/POE11#	9
MTU3_0	MTIOC0A	PB3/MTIOC0A/SCK0	32
		P31/MTIOC0A/MTCLKC/SSL1	61
	MTIOC0B	PB2/MTIOC0B/TXD0/SDA	33
		P30/MTIOC0B/MTCLKD/SSL0	63
POE	POE10#	PE2/NMI/POE10#	15
		PE4/MTCLKC/IRQ1/POE10#	8
GPT0	GTIOC0A	P71/MTIOC3B/GTIOC0A	56
	*2	PD7/GTIOC0A/CTX/SSL1/TRST#	18
	GTIOC0B	P74/MTIOC3D/GTIOC0B	53
	*2	PD6/GTIOC0B/SSL0/TMS	19
GPT1	GTIOC1A	P72/MTIOC4A/GTIOC1A	55
	*2	PD5/GTIOC1A/RXD1/TDI	20
	GTIOC1B	P75/MTIOC4C/GTIOC1B	52
	*2	PD4/GTIOC1B/SCK1/TCK	21

GPT2	GTIOC2A	P73/MTIOC4B/GTIOC2A	54
	*2	PD3/GTIOC2A/TXD1/TDO	22
	GTIOC2B	P76/MTIOC4D/GTIOC2B	51
	*2	PD2/GTIOC2B/MOSI/TRCLK	23
SCI2	TXD2	PB5/CTX/TXD2/TRSYNC	28
	*3	P81/MTIC5V/TXD2	97
	RXD2	PB6/CRX/RXD2/TRDATA0	27
	*3	P80/MTIC5W/RXD2	98
	SCK2	PB7/SCK2/TRDATA1	26
	*3	P82/MTIC5U/SCK2	96
RSPI0	RSPCK	P24/RSPCK	64
		PA4/ADTRG0#/MTIOC1B/RSPCK	37
	*4	PD0/GTIOC3B/RSPCK/TRDATA2	25
	MOSI	P23/CTX/LTX/MOSI	65
		PB0/MTIOC0D/MOSI	35
	*4	PD2/GTIOC2B/MOSI/TRCLK	23
	MISO	P22/ADTRG#/CRX/LRX/MISO	66
		PA5/ADTRG1#/MTIOC1A/MISO	36
	*4	PD1/GTIOC3A/MISO/TRDATA3	24
	SSL0	P30/MTIOC0B/MTCLKD/SSL0	63
		PA3/MTIOC2A/SSL0	38
	*4	PD6/GTIOC0B/SSL0/TMS	19
	SSL1	P31/MTIOC0A/MTCLKC/SSL1	61
		PA2/MTIOC2B/SSL1	39
	*4	PD7/GTIOC0A/CTX/SSL1/TRST#	18
	SSL2	P32/MTIOC3C/MTCLKB/SSL2	59
		PA1/MTIOC6A/SSL2	40
	*4	PE0/CRX/SSL2	17
	SSL3	P33/MTIOC3A/MTCLKA/SSL3	58
		PA0/MTIOC6C/SSL3	41
	*4	PE1/SSL3	16
S12ADA0	ADTRG0#	PA4/ADTRG0#/MTIOC1B/RSPCK	37
		P20/ADTRG0#/MTCLKB/IRQ7	68
S12ADA1	ADTRG1#	PA5/ADTRG1#/MTIOC1A/MISO	36
		P21/ADTRG1#/MTCLKA/IRQ6	67

*1 to 4 The settings are linked together

RX62G Group
Peripheral Driver Generator
Reference Manual

Publication Date: May 16, 2014 Rev.1.02

Published by: Renesas Electronics Corporation

Edited by: Microcomputer Tool Development Department 4
Renesas Solutions Corporation

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Ku, Seoul, 135-920, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

RX62G Group
Peripheral Driver Generator
Reference Manual

Renesas Electronics Corporation

R20UT2275EJ0102