User's Manual

RX63T Group

Peripheral Driver Generator Reference Manual

Renesas Electronics
www.renesas.com

Rev.1.02 May 2014

(2012.4)

Introduction

This manual was written to explain how to make the peripheral I/O drivers on the Peripheral Driver Generator for RX63T. 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 RX63T group, peripheral modules and endian.

(1) Products

The microcomputer model which has been planned and developed is included, so please confirm the status on our Web site in case of selection of a device.

Part No.	Package	Part No.	Package
R5F563TEADFB	PLQP0144KA-A	R5F563TCEDFH	PLQP0112JA-A
R5F563TEADFA	PLQP0120KA-A	R5F563TCEDFP	PLQP0100KB-A
R5F563TEADFH	PLQP0112JA-A	R5F563TBEDFB	PLQP0144KA-A
R5F563TEADFP	PLQP0100KB-A	R5F563TBEDFA	PLQP0120KA-A
R5F563TCADFB	PLQP0144KA-A	R5F563TBEDFH	PLQP0112JA-A
R5F563TCADFA	PLQP0120KA-A	R5F563TBEDFP	PLQP0100KB-A
R5F563TCADFH	PLQP0112JA-A	R5F563T6EDFM	PLQP0064KB-A
R5F563TCADFP	PLQP0100KB-A	R5F563T5EDFM	PLQP0064KB-A
R5F563TBADFB	PLQP0144KA-A	R5F563T4EDFM	PLQP0064KB-A
R5F563TBADFA	PLQP0120KA-A	R5F563T6EDFL	PLQP0048KB-A
R5F563TBADFH	PLQP0112JA-A	R5F563T5EDFL	PLQP0048KB-A
R5F563TBADFP	PLQP0100KB-A	R5F563T4EDFL	PLQP0048KB-A
R5F563TEDDFB	PLQP0144KA-A	R5F563TEAGFB	PLQP0144KA-A
R5F563TEDDFA	PLQP0120KA-A	R5F563TEAGFA	PLQP0120KA-A
R5F563TEDDFH	PLQP0112JA-A	R5F563TEAGFH	PLQP0112JA-A
R5F563TEDDFP	PLQP0100KB-A	R5F563TEAGFP	PLQP0100KB-A
R5F563TCDDFB	PLQP0144KA-A	R5F563TCAGFB	PLQP0144KA-A
R5F563TCDDFA	PLQP0120KA-A	R5F563TCAGFA	PLQP0120KA-A
R5F563TCDDFH	PLQP0112JA-A	R5F563TCAGFH	PLQP0112JA-A
R5F563TCDDFP	PLQP0100KB-A	R5F563TCAGFP	PLQP0100KB-A
R5F563TBDDFB	PLQP0144KA-A	R5F563TBAGFB	PLQP0144KA-A
R5F563TBDDFA	PLQP0120KA-A	R5F563TBAGFA	PLQP0120KA-A
R5F563TBDDFH	PLQP0112JA-A	R5F563TBAGFH	PLQP0112JA-A
R5F563TBDDFP	PLQP0100KB-A	R5F563TBAGFP	PLQP0100KB-A
R5F563TEBDFB	PLQP0144KA-A	R5F563TEBGFB	PLQP0144KA-A
R5F563TEBDFA	PLQP0120KA-A	R5F563TEBGFA	PLQP0120KA-A
R5F563TEBDFH	PLQP0112JA-A	R5F563TEBGFH	PLQP0112JA-A
R5F563TEBDFP	PLQP0100KB-A	R5F563TEBGFP	PLQP0100KB-A
R5F563TCBDFB	PLQP0144KA-A	R5F563TCBGFB	PLQP0144KA-A
R5F563TCBDFA	PLQP0120KA-A	R5F563TCBGFA	PLQP0120KA-A
R5F563TCBDFH	PLQP0112JA-A	R5F563TCBGFH	PLQP0112JA-A
R5F563TCBDFP	PLQP0100KB-A	R5F563TCBGFP	PLQP0100KB-A
R5F563TBBDFB	PLQP0144KA-A	R5F563TBBGFB	PLQP0144KA-A
R5F563TBBDFA	PLQP0120KA-A	R5F563TBBGFA	PLQP0120KA-A
R5F563TBBDFH	PLQP0112JA-A	R5F563TBBGFH	PLQP0112JA-A
R5F563TBBDFP	PLQP0100KB-A	R5F563TBBGFP	PLQP0100KB-A
R5F563TEEDFB	PLQP0144KA-A	R5F563T6EGFM	PLQP0064KB-A
R5F563TEEDFA	PLQP0120KA-A	R5F563T5EGFM	PLQP0064KB-A
R5F563TEEDFH	PLQP0112JA-A	R5F563T4EGFM	PLQP0064KB-A
R5F563TEEDFP	PLQP0100KB-A	R5F563T6EGFL	PLQP0048KB-A
R5F563TCEDFB	PLQP0144KA-A	R5F563T5EGFL	PLQP0048KB-A
R5F563TCEDFA	PLQP0120KA-A	R5F563T4EGFL	PLQP0048KB-A

(2) Peripheral Modules

Voltage Detection Circuit (LVDA)	General PWM Timer (GPT)
Clock Generation Circuit	Compare Match Timer (CMT)
Clock Frequency Accuracy Measurement Circuit (CAC)	Watchdog Timer (WDTA)
Low Power Consumption	Independent Watchdog Timer (IWDTa)
Register Write Protection Function	Serial Communications Interface (SCIc,SCId)
Exceptions, Interrupt Controller (ICUb)	I^2C Bus Interface (RIIC)
Buses	Serial Peripheral Interface (RSPI)
DMA Controller (DMACA)	CRC Calculator (CRC)
Data Transfer Controller (DTCa)	12-Bit A/D Converter (S12ADB)
I/O Ports	10-Bit A/D Converter (AD)

Multifunction Pin Controller (MPC)	D/A Converter (DAa)
Multi-Function Timer Pulse Unit 3 (MTU3)	Data Operation Circuit (DOC)
Port Output Enable 3 (POE3)	

(3) Endian

Little and Big

1.2 Tool requirements

The following tools are required for this version of RX63T group Peripheral Driver Generator.

- RX Family C/C++ Compiler Package V.1.02 Release 01
- RX63T Group Renesas Peripheral Driver Library V.2. 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.

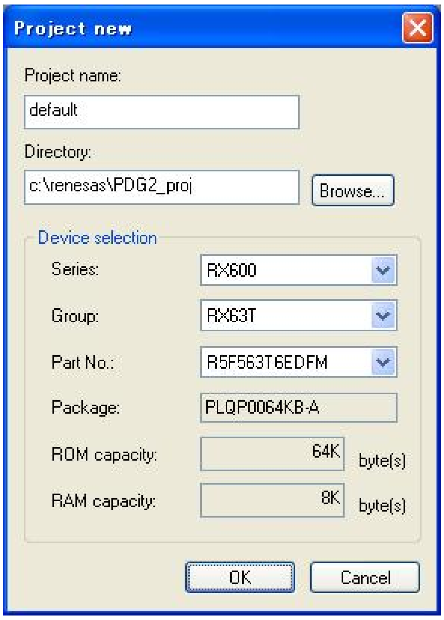


Fig 2.1 New project dialog box

For RX63T group, select [RX600] as a series and select [RX63T] 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.

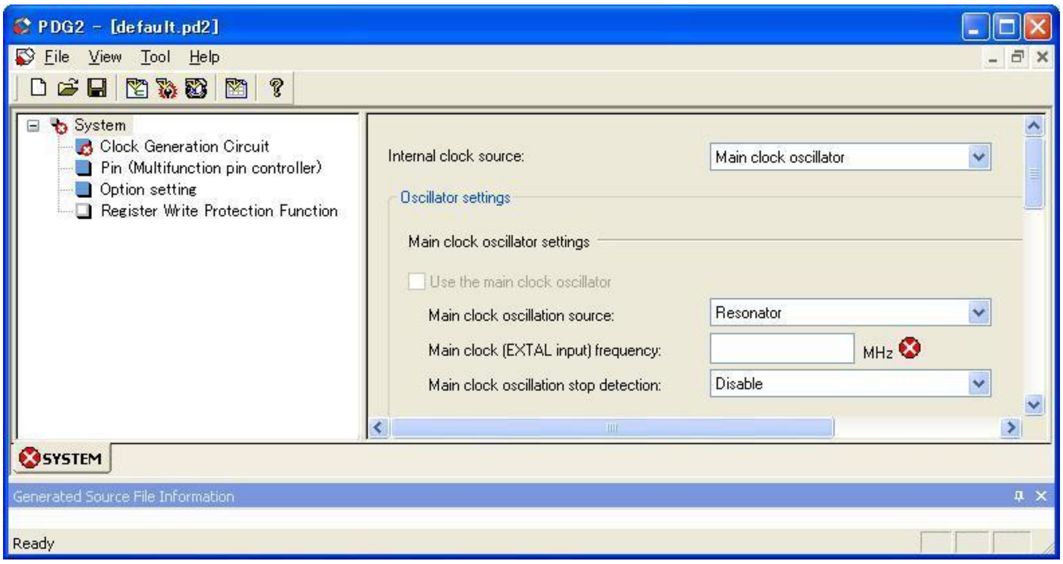


Fig 2.2 Error display of new project

Set the frequency of the clock to be used here.

Each value (e.g. frequency) entered in the window will be rounded to its nearest valid value after division or multiplication. The final value is displayed as "Actual value" on the GUI.

3. Setting Up the Peripheral Modules

3.1 Main Window

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

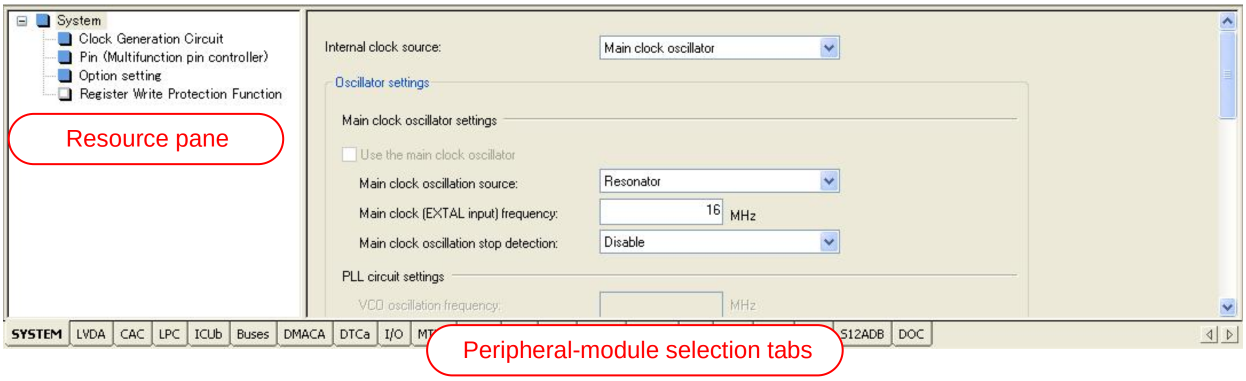


Figure 3.1 Display in the Main Window (Example)

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(Multifunction pin controller)	Pinfunctions (Multifunction Pin Controller (MPC))
	Option setting	Endian setting
	Register Write Protection Function	Register Write Protection Function
LVDA	Voltage monitoring 0 to 2	Voltage monitoring 0 to 2
CAC	Clock frequency accuracy measurement circuit (CAC)	Clock Frequency Accuracy Measurement Circuit (CAC)
LPC	Low Power Consumption	Low Power Consumption
ICUb	Interrupts	Interrupt Control Unit (ICUb) (Fastinterrupt, Software Interrupt, External Interrupt (NMI, IRQ0 to IRQ7))
	Exceptions	Exceptions
Buses	Common settings	Bus Priority and Bus Error Monitoring
DMACA	DMAC0 to DMAC3	DMA Controller (DMACA) Channel 0 to 3
DTCa	Data transfer controller (DTCa)	Data Transfer Controller (DTCa)
I/O	Port 0 to 9, A to G	I/O Port 0 to 9, A to G
MTU3	MTU0 to MTU7	Multi-Function Timer Pulse Unit 3 (MTU3) Channel 0 to 7
POE3	Port Output Enable 3 (POE3)	Port Output Enable 3 (POE3)
GPT	GPT0 to GPT3	General PWM Timer (GPT) Channel 0 to 3
CMT	Unit0 (CMT0 and CMT1)	Compare Match Timer (CMT) Unit 0 (Channel 0 and 1)
	Unit1 (CMT2 and CMT3)	Compare Match Timer (CMT) Unit 1 (Channel 2 and 3)
WDTA	Watchdog Timer (WDTA)	Watchdog Timer (WDTA)
IWDTa	Independent Watchdog Timer (IWDTa)	Independent Watchdog Timer (IWDTa)

SCI	SCI0 to 3 and 12	Serial Communications Interface SCIc(SCI0 to 3) and SCId(SCI12)
RIIC	RIIC0 and RIIC1	I^2C Bus Interface (RIIC) Channel 0 and 1
RSPI	RSPI0 and RSPI1	Serial Peripheral Interface (RSPI) Channel 0 and 1
CRC	CRC Calculator (CRC)	CRC Calculator (CRC)
S12ADB	S12AD0	12-Bit A/D Converter (S12ADB)
	Comparator	Comparator
AD	AD0	10-Bit A/D Converter (AD)
DAa	DA0 and DA1	D/A Converter (DAa) Channel 0 and 1
DOC	Data Operation Circuit (DOC)	Data Operation Circuit (DOC)

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 (Multifunction Pin Controller)

The multifunction pin controller (MPC) in RX63T-group MCUs selects the functions to be assigned to individual pins. The PDG provides a pin-function pane through which settings for the MPC can be made. Select the [SYSTEM] tab from the peripheral-module selection tabs and click on [Pin (Multi function pin controller)] 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. The two sheets are linked, so that settings can be made in either of them.

3.2.1 [Pin function] Sheet

(1) Configuration

The [Pin function] sheet shows all of the MCU pins in order and the functions that have been assigned to those pins. This sheet can be used to select functions for each of the pins with multiplexed functions.

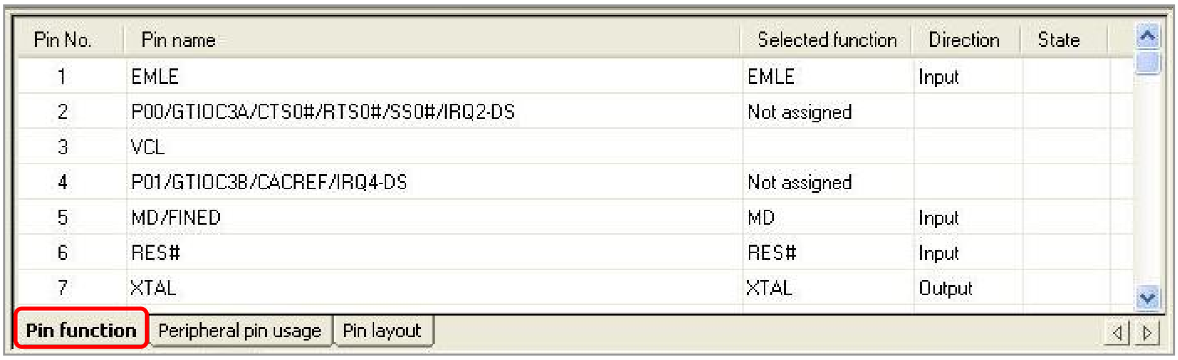


Figure 3.3 Pin-Function Pane ([Pin function] Sheet)

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	Currently allocated pin function
Direction	Whether the pin function is an input or output
State	Warning or error message, if any

(2) Default State

By default (i.e. when no pins have been set up for use with peripheral modules), “Not assigned” is shown in the [Selected function] column for each port pin, indicating that no function has yet been selected (figure 3.4).

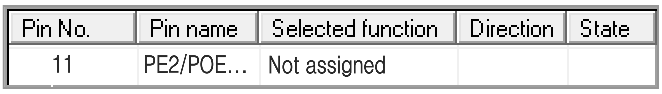
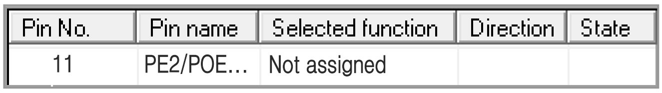


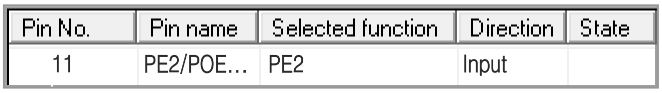
Figure 3.4 [Pin function] Sheet in the Default State (64-Pin LQFP Package)

Note:

Port pins of RX63T-group MCUs are general-purpose input port pins by default. Even though “Not assigned” is shown in the [Selected function] column for each port pin by default (i.e. when no pins have been set up for use with peripheral modules), the pin will act as a general-purpose input port pin. When you designate a pin as a general-purpose input port pin in the [I/O] pane, the name of the general-purpose input port pin will appear in the [Selected function] column (figure 3.5(b)).



(a) Default State



(b) After Designating PE2 as a General-Purpose Input Port Pin in the [I/O] Pane

Figure 3.5 Display for Pin No.11 (64-Pin Package)

(3) Selecting a Pin Function

When a pin has multiplexed functions, placing the mouse pointer on the [Selected function] column in the row for that pin brings up a drop-down button. Clicking on the button brings up a list of selectable pin functions (figure 3.6).

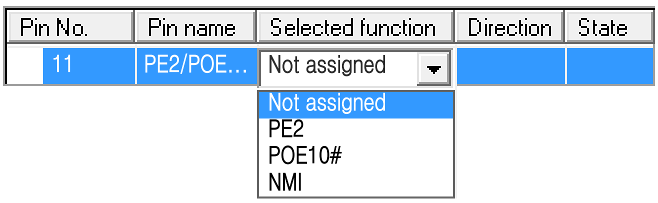


Figure 3.6 Selectable Pin Functions

In the default state (i.e. when no pins have been set up for use with peripheral modules), if [Selected function] is changed from “Not assigned” to another pin function, the warning [<Name of the pin function> has not been configured in the peripheral settings.] appears. For example, when [Selected function] for PE2/POE10#/NMI is changed from “Not assigned” to NMI despite the interrupt controller (ICUb) not being set up, a warning appears as shown in figure 3.7.

Pin No.	Pin name	Selected function	Direction	State
11	PE2/POE...	NMI		NMI has not been configured in the peripheral settings.

Figure 3.7 Warning on Changing [Selected function] in the Default State

When the NMI has been set up in the [ICUb] pane, the warning disappears and “NMI” appears in the [Selected function] column.

Pin No.	Pin name	Selected function	Direction	State
11	PE2/POE...	NMI	Input	

Figure 3.8 After Setting the NMI up

Note:
The generation of source files is still possible when the warning shown in figure 3.7 is being displayed, but the pin will not act as an NMI. For details, refer to section 3.2.5, Error Messages and Warnings on Pin Settings.

(4) Selecting a Pin Function before Setting up the Associated Peripheral Module
When a peripheral module is set up after selecting the pin functions on the [Pin function] sheet, the selected pin functions are automatically allocated to the pins.
IRQ1, for example, can be assigned to P93or P11. To assign IRQ1 to P93, IRQ1 should be selected as the [Selected function] for P93 on the [Pin function] sheet (figure 3.9).

Pin No.	Pin name	Selected function	Direction	State
30	P93/IRQ1/...	IRQ1		IRQ1 has not been configured in the peripheral settings.

Figure 3.9 IRQ1 Selected for P93 (with the ICUb Not Set up)

When IRQ1 is set up in the [ICUb] pane, IRQ1 is actually assigned to P93 (figure 3.10).

Pin No.	Pin name	Selected function	Direction	State
30	P93/IRQ1/...	IRQ1	Input	

Figure 3.10 IRQ1 Selected for P93 (after the ICUb Has been Set up)

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 associated with the peripheral module selected in the left section and where those functions are assigned are listed in the right section. If multiple pins are selectable for a specific function, the allocation can be changed through this sheet.

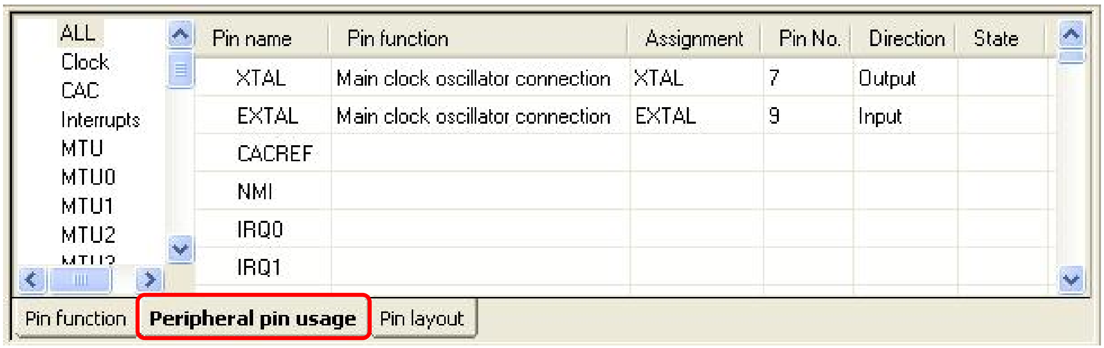


Figure 3.11 Pin-Function Pane ([Peripheral pin usage] Sheet)

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	Warning or error message, if any

(1) Default State

By default (i.e. when no pins have been set up for use with peripheral modules), the [Pin Function] and [Assignment] columns are blank (figure 3.12).

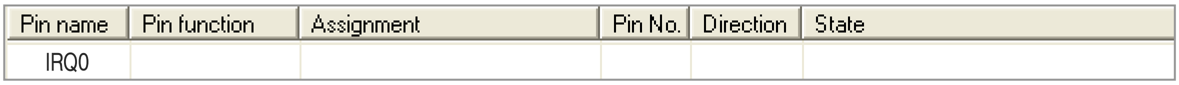


Figure 3.12 [Peripheral pin usage] Sheet in the Default State

(2) Assigning a Pin Function to a Port Pin

When a peripheral module associated with input to or output from pins has been set up, the pin functions to be used by that peripheral module are assigned to the corresponding port pins and the current settings are shown on the [Peripheral pin usage] sheet. If you have set up external interrupt IRQ0 in the detailed settings pane, for example, pin IRQ0 is assigned to P10 and the [Peripheral pin usage] sheet shows the setting of IRQ0 as follows.

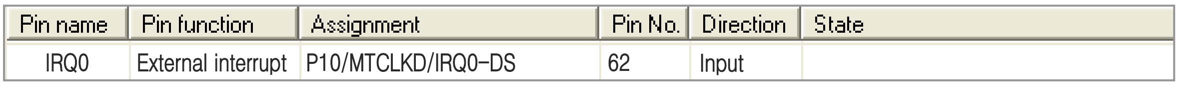


Figure 3.13 Display of a Pin Function Assigned to a Port Pin (Example)

Note:
When a peripheral module is set up in the default state (i.e. when no pin functions have been selected on the [Pin function] or [Peripheral pin usage] sheet), the pin functions for that peripheral module are assigned to the port pins listed in the “Allocation in the Default State” section of appendix 1, Pin Functions for which the Allocation Can be Changed. When the allocation of pin functions has been designated on the [Pin function] sheet before a peripheral module is set up, the pin functions are assigned to the selected port pins.

Subsequently setting up general-purpose I/O port pin P10, which uses the same pin as IRQ0, in the [I/O] pane will cause a conflict and a warning will be output as shown in figure 3.14.

Pin name	Pin function	Assignment	Pin No.	Direction	State
IRQ0	External interrupt	P10/MTCLKD/IRQ0-DS	62	Input	Conflicting with another pin function.

Figure 3.14 Warning of a Conflict between Pin Functions

Note:
Even if two or more pin functions are assigned to a single pin (as in figure 3.14), generating source files is still possible. You can switch between the functions, although more than one cannot be in use at the same time. For details, refer to section 3.2.5, Error Messages and Warnings on Pin Settings.

The allocation of IRQ0 can be changed. 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/IRQ0-DS	62	Input	Conflicting with another pin function.

Figure 3.15 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/IRQ0-DS	62	Input	Conflicting with another pin function.
		P10/MTCLKD/IRQ0-DS			
		PB5/POE11#/TXD12/SMOSI12/SSDA12/TXDX12/SIOX12/IRQ0			

Figure 3.16 Changing the Allocation of a Pin Function

If IRQ0 is assigned to PB5 and that pin is not being used for 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	PB5/POE11#/TXD12/...	19	Input	

Figure 3.17 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.

Note:
When the peripheral module has not been set up (as in figure 3.12), the allocation of pin functions cannot be changed through this sheet.

3.2.3 [Pin layout] Sheet

(1) Configuration

The [Pin layout] sheet shows graphical pin layout view. This sheet can be used to select functions for each of the pins with multiplexed functions.

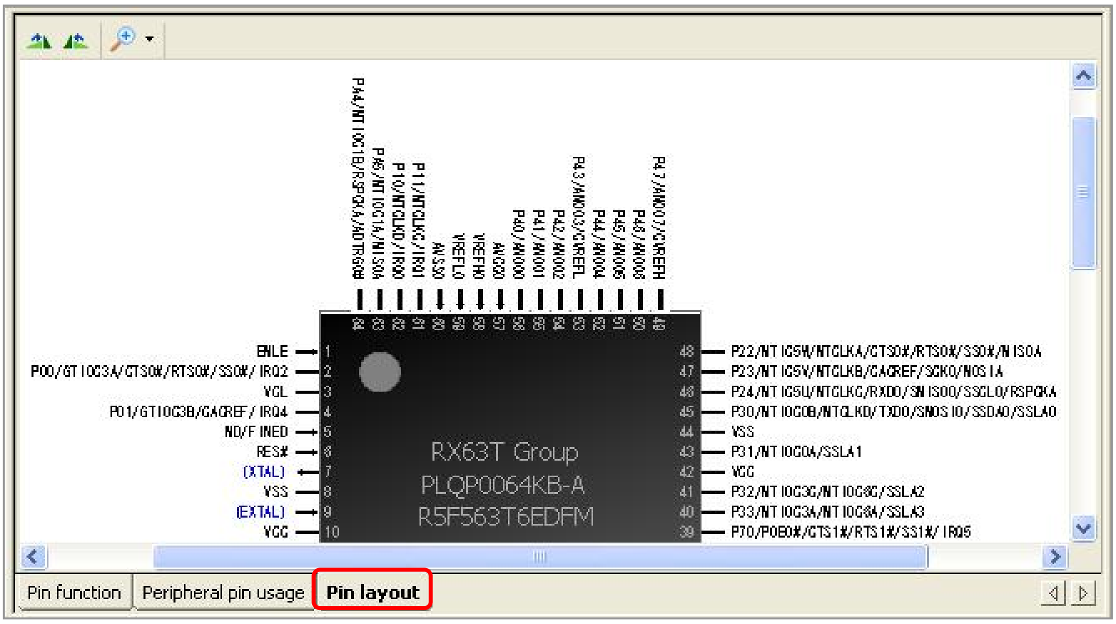


Figure 3.18 Pin-Function Pane ([Pin layout] Sheet)

(2) Functions

The [Pin layout] sheet has the following functions.

- Rotate

The [Rotate] buttons ( ) rotate the view by 90 degrees clockwise or counter clockwise.

- Zoom In/Zoom out

The [Zoom in] button ( ) zooms the view by an additional 25%. It also has a drop-down list of zoom level.

(3) Selecting a Pin Function

Placing the mouse pointer on the pin which has multiplexed functions and clicking right button brings up a list of selectable pin functions. (Figure 3.20)

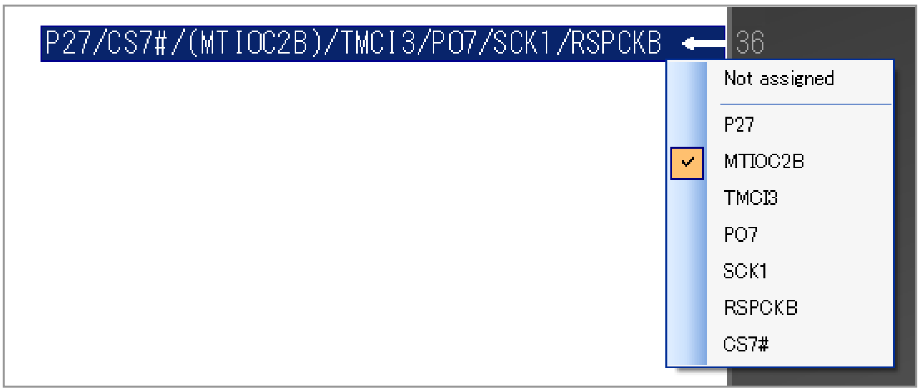


Figure 3.20 Pin function selection

The selection of pin function can be changed from this list. Setting changes on [Pin layout] sheet are reflected on the other sheets. For details, refer to 3.2.4 Peripheral-Module Setting Shared byxxx.

(4) Pin Status Display

The status of each pin are displayed as follows.

- Selected function

If the pin function is assigned to the pin, the selected pin function is indicated by brackets as shown in Figure.3.21.

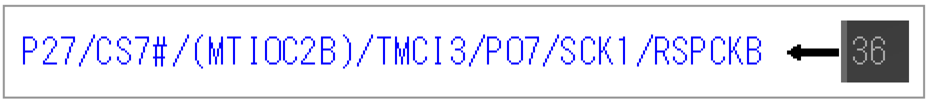


Figure 3.21 Indication of selected function (In the case when MTIOC2B is selected)

- Input/Output Direction

The signal direction of selected pin function is displayed as shown in Figure 3.22.

a. Pin function is not assigned b. Output c. Input d. Input/Output

Figure 3.22 Display of input/output direction

Note:

If two or more pin functions are assigned to one pin, signal direction is not shown.

- Error or Warning Status

The setting status of each pin is displayed as shown in Figure 3.23.

a. Pin function is not assigned (indicated by red characters)

b. Pin function is assigned and no error or warning is detected (indicated by blue characters)

c. Pin function is assigned and a warning is detected (indicated by brown characters)

d. Pin function is assigned and an error is detected (indicated by red characters)

図 3.23 Display of error or warning status

For the contents of error or warning, refer to the corresponding pin in [Pin function] sheet. For the details of error or warning in pin function window, refer to the section 3.2.5 Error Messages and Warnings on Pin Settings.

3.2.4 Pin Settings Shared between Setting Windows

A change to a setting on either the [Pin function] or [Peripheral pin usage] sheet is reflected on the other sheet. When the allocation of a pin function is changed on the [Pin function] sheet, that change also applies to the [Peripheral pin usage] sheet, and vice versa (figure 3.24).

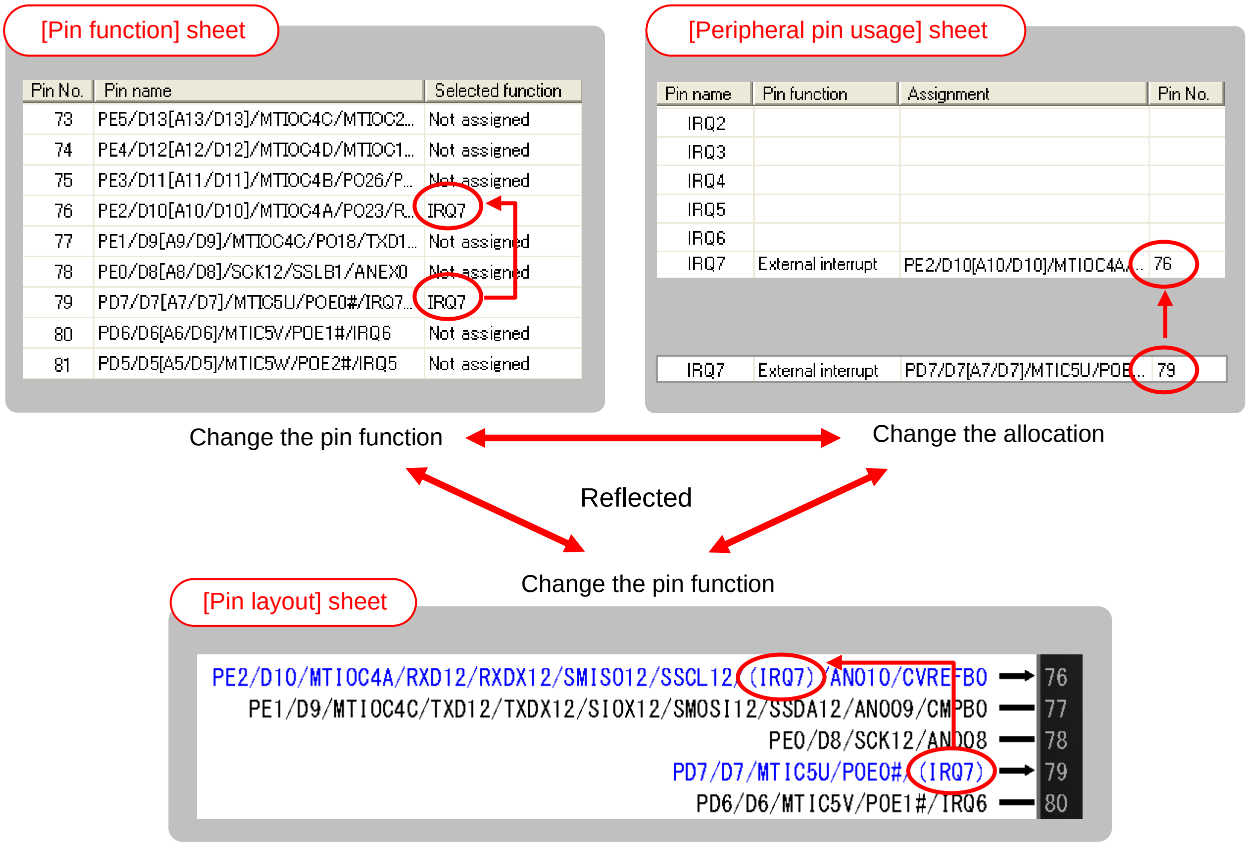


Figure 3.24 Linking of the [Pin function],[Pin layout] and [Peripheral pin usage] Sheets

The current settings for each peripheral module are reflected on the [Pin function] and [Peripheral pin usage] sheets. When IRQn is set up in the [ICUb] pane, for example, the [Peripheral pin usage] sheet shows that IRQn is in use and the allocation of IRQn is displayed on the [Pin function] and [Peripheral pin usage] sheets.

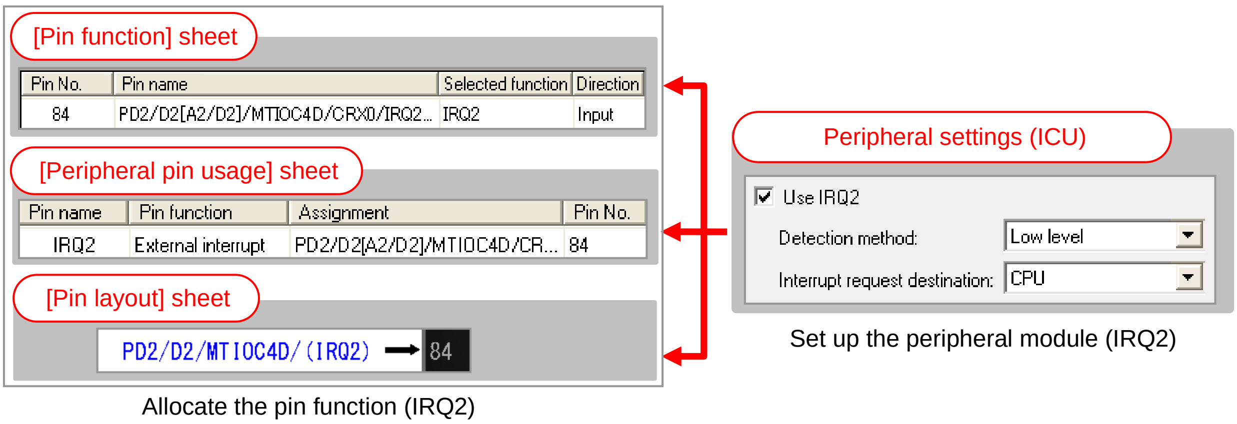


Figure 3.25 Setting up a Peripheral Module and Allocating Pin Functionss

When the setting for IRQn in the [ICUb] pane is canceled, the allocation of IRQn is canceled on the [Pin function] and [Peripheral pin usage] sheets.

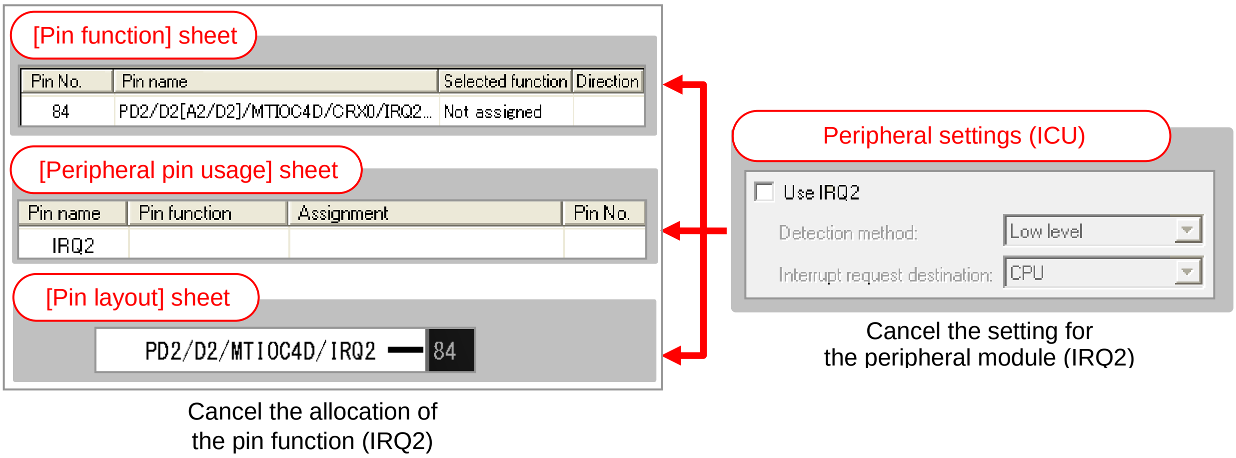


Figure 3.26 Deleting setting of Peripheral Module and Deallocating Pin Functions

On the other hand, a change made on the [Pin function] or [Peripheral pin usage] sheet is not reflected on the detailed-settings pane for the peripheral module. Even if [Selected function] for IRQn is changed to “Not assigned” on the [Pin function] sheet (or [Pin layout] sheet) after IRQn has been set up in the [ICUb] pane, for example, the setting of IRQn in the [ICUb] pane is not canceled. Since no pin is assigned to IRQn in this case, an error message appears. For details on the error messages, refer to section 3.2.5, Error Messages and Warnings on Pin Settings.

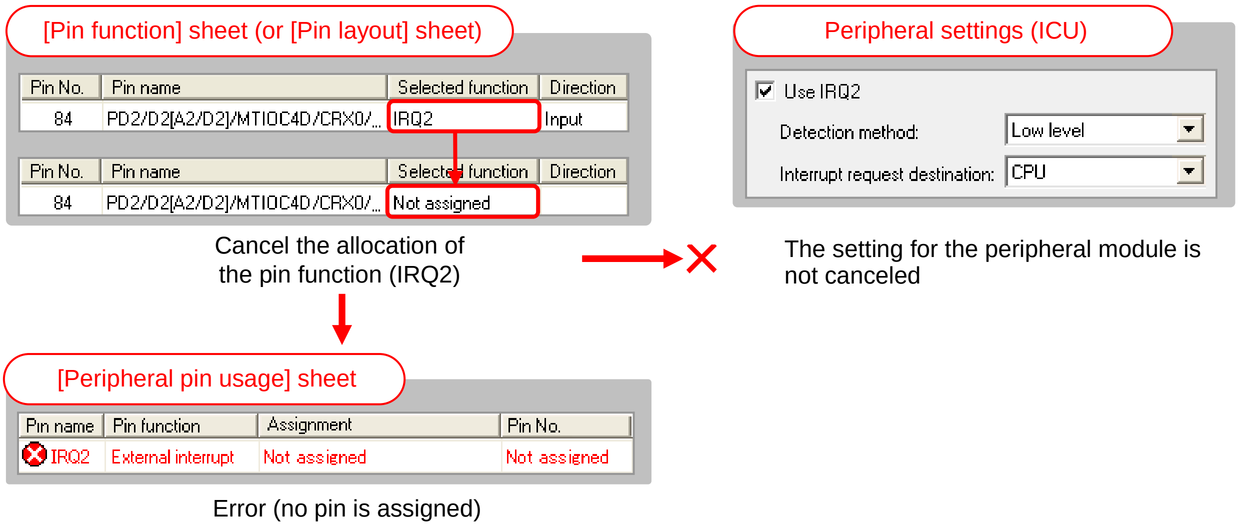


Figure 3.27 Canceling the Allocation of a Pin Function Leading to Display of an Error Message

3.2.5 Error Messages and Warnings on Pin Settings

When an incorrect setting is made, an error message or warning is displayed on the [Pin function] or [Peripheral pin usage] sheet. The errors and warnings are listed in table 3.4.

Table 3.4 Errors and Warnings

Cause	Type	Message
A single pin function has been selected for multiple pins.	Error	"The same function is assigned to *<pin numbers>*." ([Pin function] sheet) "Do not assign a single function to multiple pins." ([Peripheral pin usage] sheet)
The pin function has not been allocated.	Error	"Not assigned" ([Peripheral pin usage] sheet)
Multiple pin functions have been selected for a single pin.	Warning	"Conflicting between different functions." ([Pin function] sheet) "Conflicting with another pin function." ([Peripheral pin usage] sheet)
Conflict with use of a pin by a debugger	Warning	"Conflicting with an on-chip emulator pin." ([Pin function] sheet) "Conflicting between a peripheral module pin and an on-chip emulator pin." ([Peripheral pin usage] sheet)
The peripheral module has not been set up.	Warning	"*<pin function>* has not been configured in the peripheral settings." ([Pin function] sheet)

Details of the errors and warnings are given below.

(1) A single pin function has been selected for multiple pins.

Selecting a single pin function for multiple pins leads to an error that prevents the generation of source files. In this case, allocate another pin function to either of the pins, change the entry on the [Pin function] sheet to "Not assigned", or re-select the allocation of the pin function on the [Peripheral pin usage] sheet.

Pin No.	Pin name	Selected function	Direction	State
18	P32/MTIOC0C/TIOCC0/TMO3/PO10/RTCOUT/RTC..	IRQ2	Input	The same function is assigned to 18/84.
84	PD2/D2[A2/D2]/MTIOC4D/CRX0/IRQ2/AN010	IRQ2	Input	The same function is assigned to 18/84.

(a) [Pin function] Sheet

Pin name	Pin function	Assignment	Pin No.	Direction	State
IRQ2	External interrupt	Conflicted	18/84	Input	Do not assign a single function to multiple pins.

(b) [Peripheral pin usage] Sheet

Figure 3.28 Example of an Error (Selection of a Single Function for Multiple Pins)

(2) The pin function has not been allocated.

Failure to allocate a pin function required by a peripheral module leads to an error and prevents the generation of source files.Select the pin function for a corresponding pin on the [Pin function] sheet or designate the allocation of the pin function on the [Peripheral pin usage] sheet.

Pin name	Pin function	Assignment	Pin No.	Direction	State
IRQ2	External interrupt	Not assigned	Not assigned	Input	Not assigned.

[Peripheral pin usage] Sheet

Figure 3.29 Example of an Error (Pin Function not Allocated)

(3) Multiple pin functions have been selected for a single pin.

A warning appears when two or more pin functions have been assigned to a single pin (as in figure 3.30), but generating source files is still possible. You can switch between the functions, although they cannot be used at the same time.
To switch between pin functions, make the initial setting for the peripheral module using that pin function, since the individual pin functions are set by the initial-setting function for the given peripheral module.

Pin No.	Pin name	Selected function	Direction	State
18	P32/MTIOC0C/TIOCC0/TMO3/PO10/RTCOUT/RTC..	P32/IRQ2		Conflicting between different functions.

(a) [Pin function] Sheet

Pin name	Pin function	Assignment	Pin No.	Direction	State
IRQ2	External interrupt	P32/MTIOC0C/TIOCC0/TMO3/..	18	Input	Conflicting with another pin function.

Pin name	Pin function	Assignment	Pin No.	Direction	State
P32	General input port	P32/MTIOC0C/TIOCC0/TMO3...	18	Input	Conflicting with another pin function.

(b) [Peripheral pin usage] Sheet

Figure 3.30 Example of a Warning (Multiple Pin Functions Selected for a Single Pin)

(4) Conflict with use of a pin by a debugger

A warning appears when a pin function for a peripheral module has been allocated to a pin for use by an on-chip debugger. Generating source files is still possible. Note, however, that the other pin function allocated to the pin may not be usable while the on-chip debugger is in use.

Pin No.	Pin name	Selected function	Direction	State
20	TDI/P30/MTIOC4B/TMRI3/PO8/RTCIC0/POE8#/R...	IRQ0		Conflicting with an on-chip emulator pin.

(a) [Pin function] Sheet

Pin name	Pin function	Assignment	Pin No.	Direction	State
IRQ0	External interrupt	TDI/P30/M...	20	Input	Conflicting between a peripheral module pin and an on-chip emulator pin.

(b) [Peripheral pin usage] Sheet

Figure 3.31 Example of a Warning (Conflict with Use of a Pin by a Debugger)

(5) The peripheral module has not been set up.

A warning appears when a pin function is selected on the [Pin function] sheet but the corresponding peripheral module has not been set up. Although generating source files is still possible, the selected pin function will not be usable. To enable the selected pin function, set up the peripheral module that is to use the function and call the initial-setting function, which sets the registers to change the pin function.

Pin No.	Pin name	Selected function	Direction	State
73	PE5/D13[..	IRQ5		IRQ5 has not been configured in the peripheral settings.

[Pin function] Sheet

Figure 3.32 Example of a Warning (Peripheral Module Not Set up)

3.3 Endian

Select the [SYSTEM] tab from the peripheral-module selection tabs and click on [Option setting] in the resource pane to open the endian setting pane.

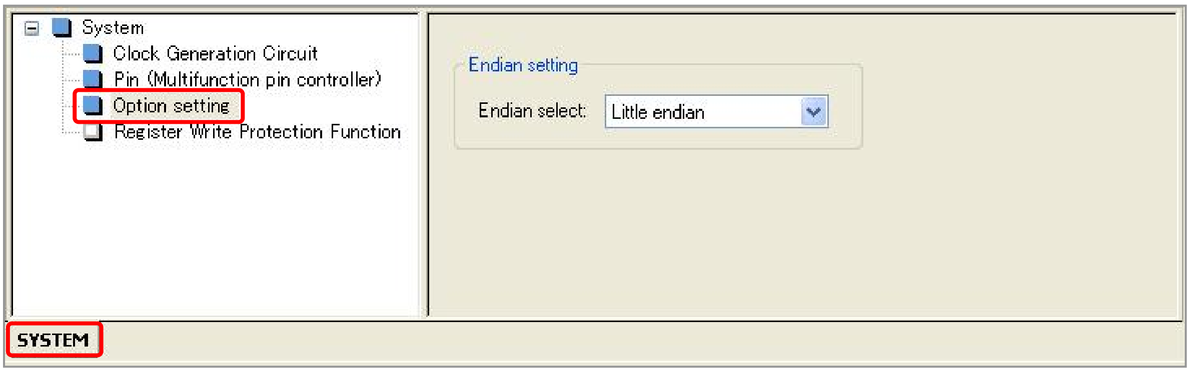


Figure 3.33 The setting method of endian

Select endian to be used here. This setting is only used for selecting RPDL library files (xxx_little.lib or xxx_big.lib) to be linked and thus does not affect the output source code.

4. Tutorial

4.1 When the HEW is in Use with Renesas starter kit for RX63T(64-pin)

This section introduces the usage of the PDG by giving instructions on how to use the PDG and HEW to create a tutorial program that implements the following operations on the Renesas Starter Kit board for the RX63T(64-pin).

・ An LED blinking on a PWM output of the multi-function timer pulse unit 3 (MTU3)
・ Continuously scanning on 12-Bit A/D converter (S12ADB)
・ Triggering DTCa by ICUb
・ Data transfer between SCIc channels 0 and 1

The labels given below respectively indicate operations to take place in the PDG and in the HEW.

PDG	: Operations in the PDG
HEW	: Operations in the HEW

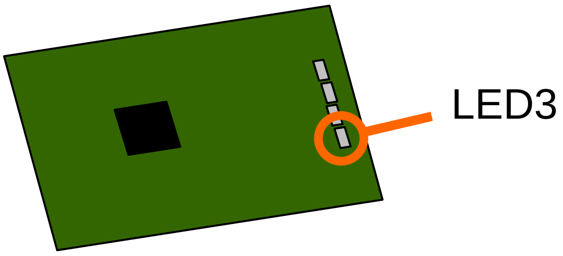
[Note on Using the HEW]
Refer to the user’s manual and check the HewTargetServer settings.

4.1.1 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.

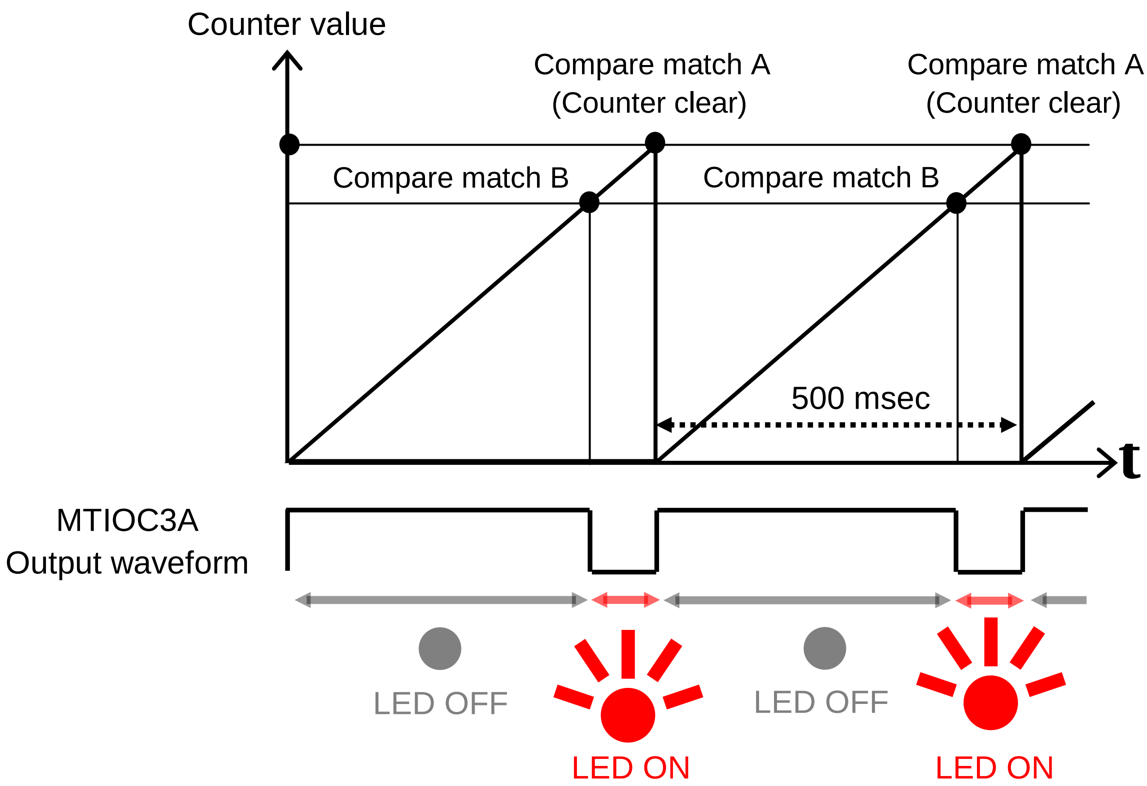
Note : If there is a switch that enables/disables P33(MTIOC3A) on the RSK board, enable it.

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



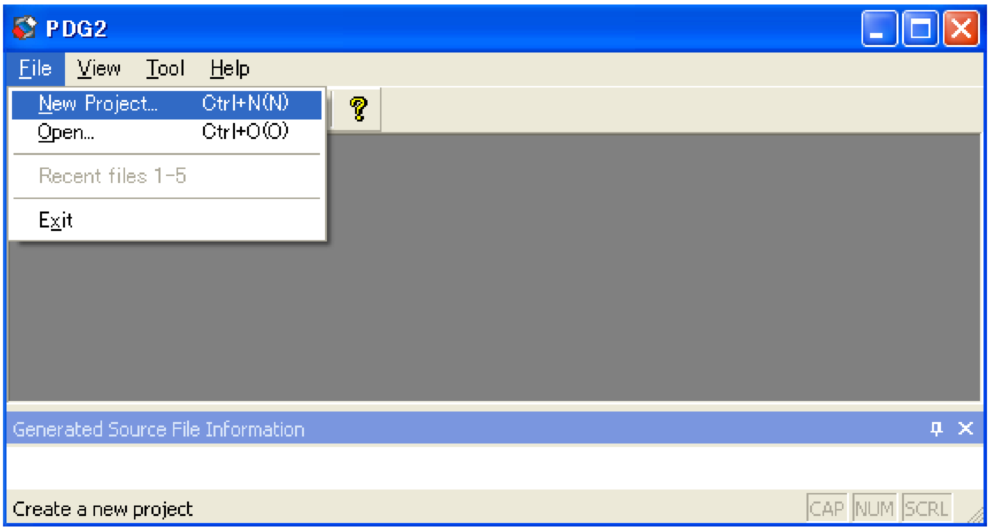
The MTU3 channel 3 (MTU3) will be operated in PWM mode 1. In PWM mode 1, the output signal is controlled by compare match A and B.

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 500 msec)



(1) Making the PDG project **PDG**

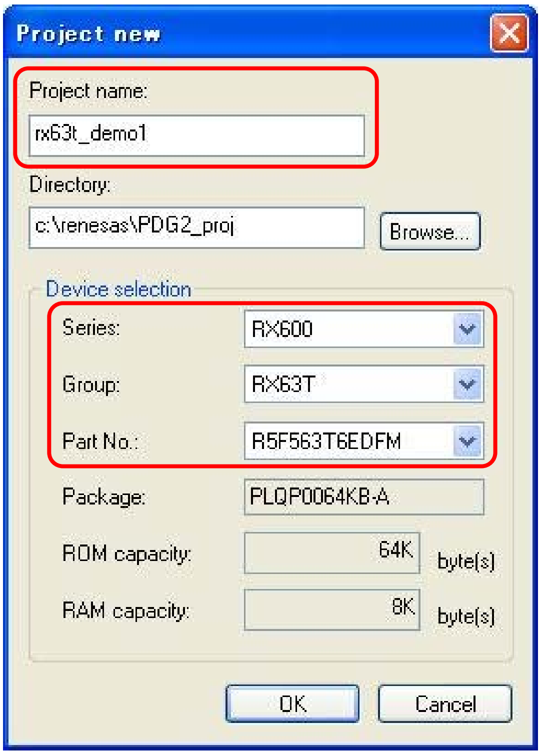
1. Start the PDG.
2. Select [File]->[New Project] menu.



3. Specify "rx63t_demo1" as the project name.
   Set the CPU type as follows.

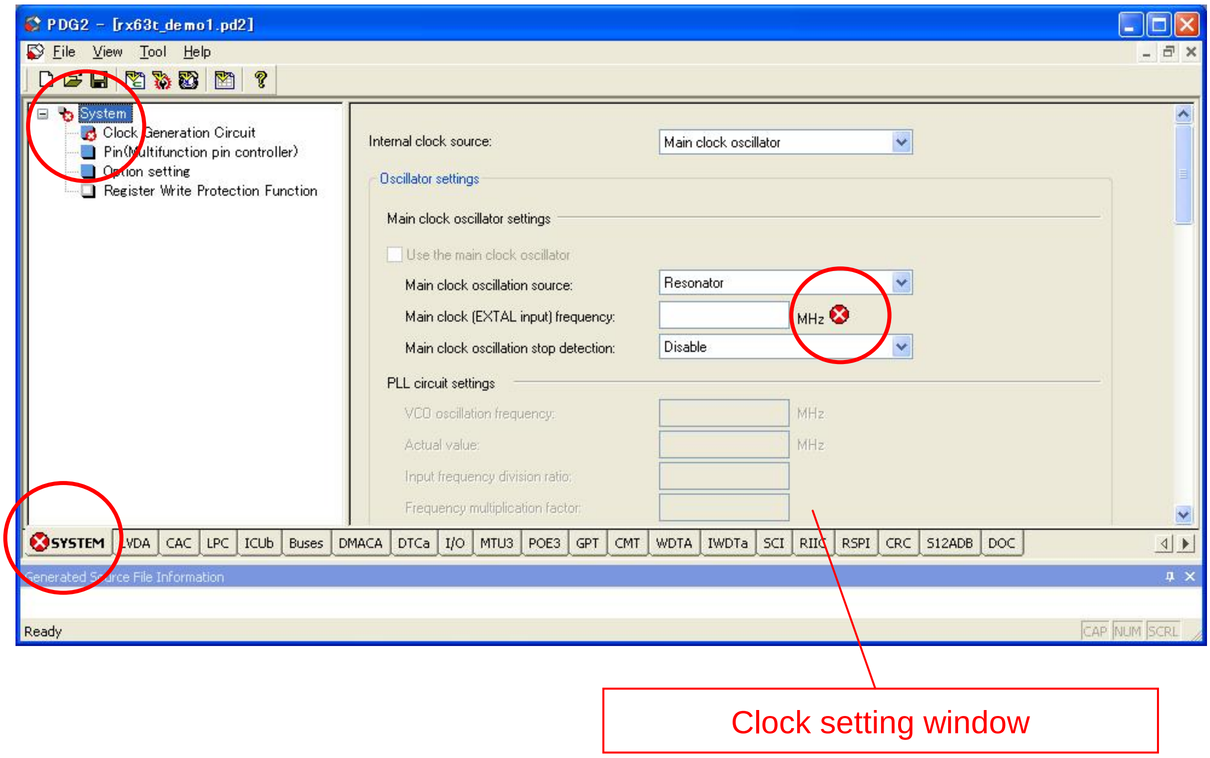
   Series : RX600
   Group : RX63T
   Part No. : R5F563T6EDFM

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

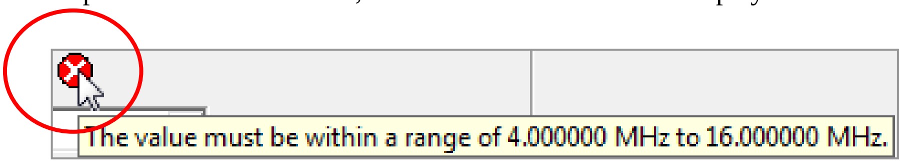


(2) Initial state **PDG**

-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 PDG

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

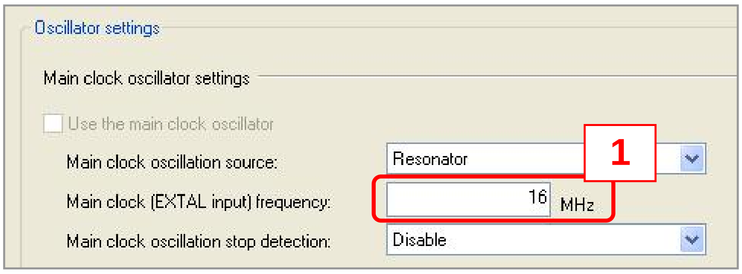
Warning
- The setting is possible but may be wrong.
- Source files can be generated.

Information
- Additional information for the complex setting.

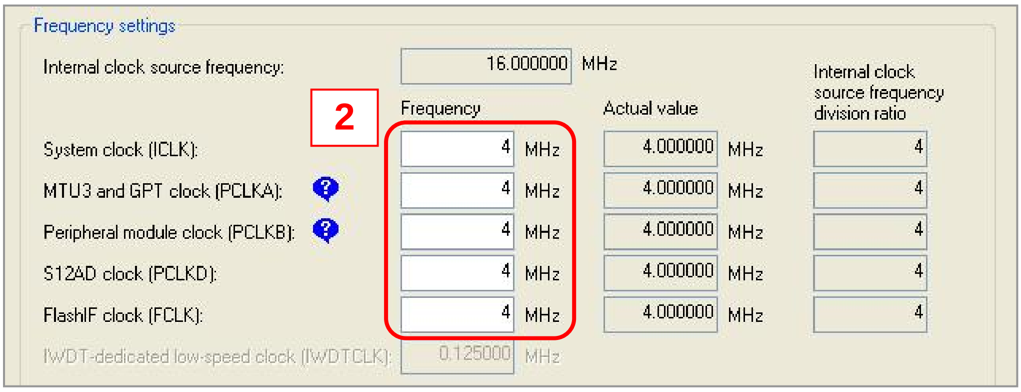
Only icons on the setting window can display the tooltip.

(3) Clock setting **PDG**

1. It is necessary to set the main (EXTAL) clock frequency first.
   External clock frequency of the RSK board is 16 MHz. Set 16 to the edit box.



2. ICLK, PCLKA, PCLKB, PCLKD and FCLK are used in 4 MHz.
   Set 4 to the edit box.



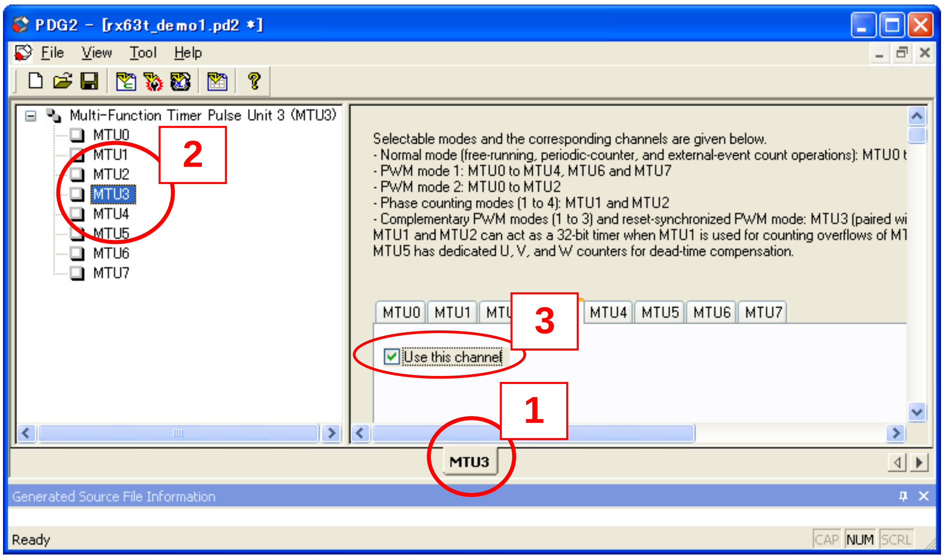
(4) Endian setting **PDG**

For the endian setting, refer to section 3.3, Endian.

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

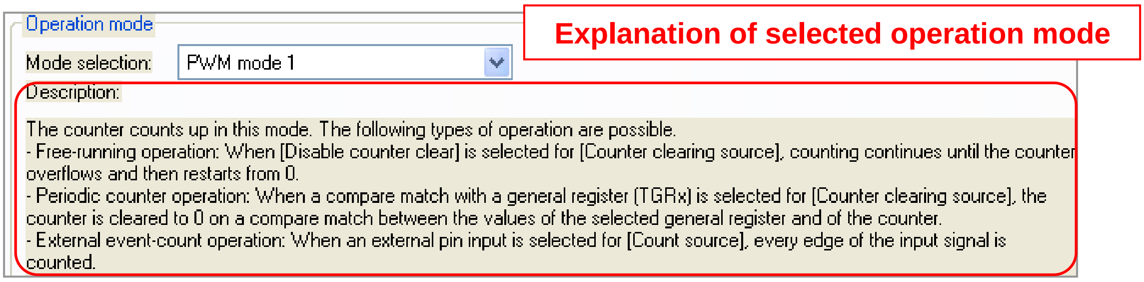
Opening MTU3 channel 3(MTU3) setting window.

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



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

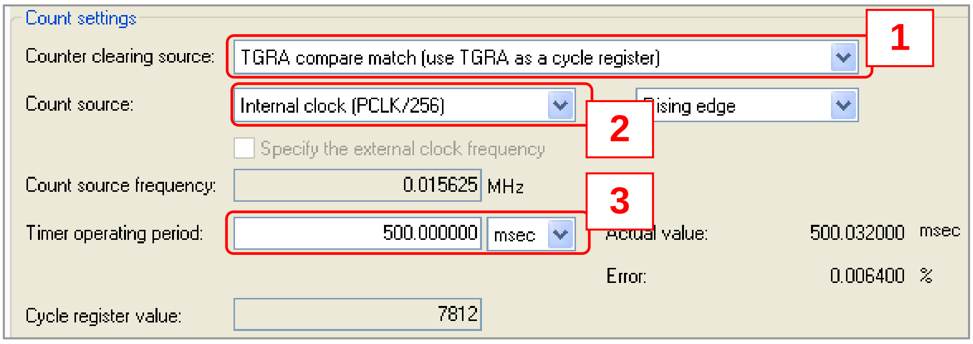
Select “PWM mode 1” for the operation mode.



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

The counter setting is as follows.

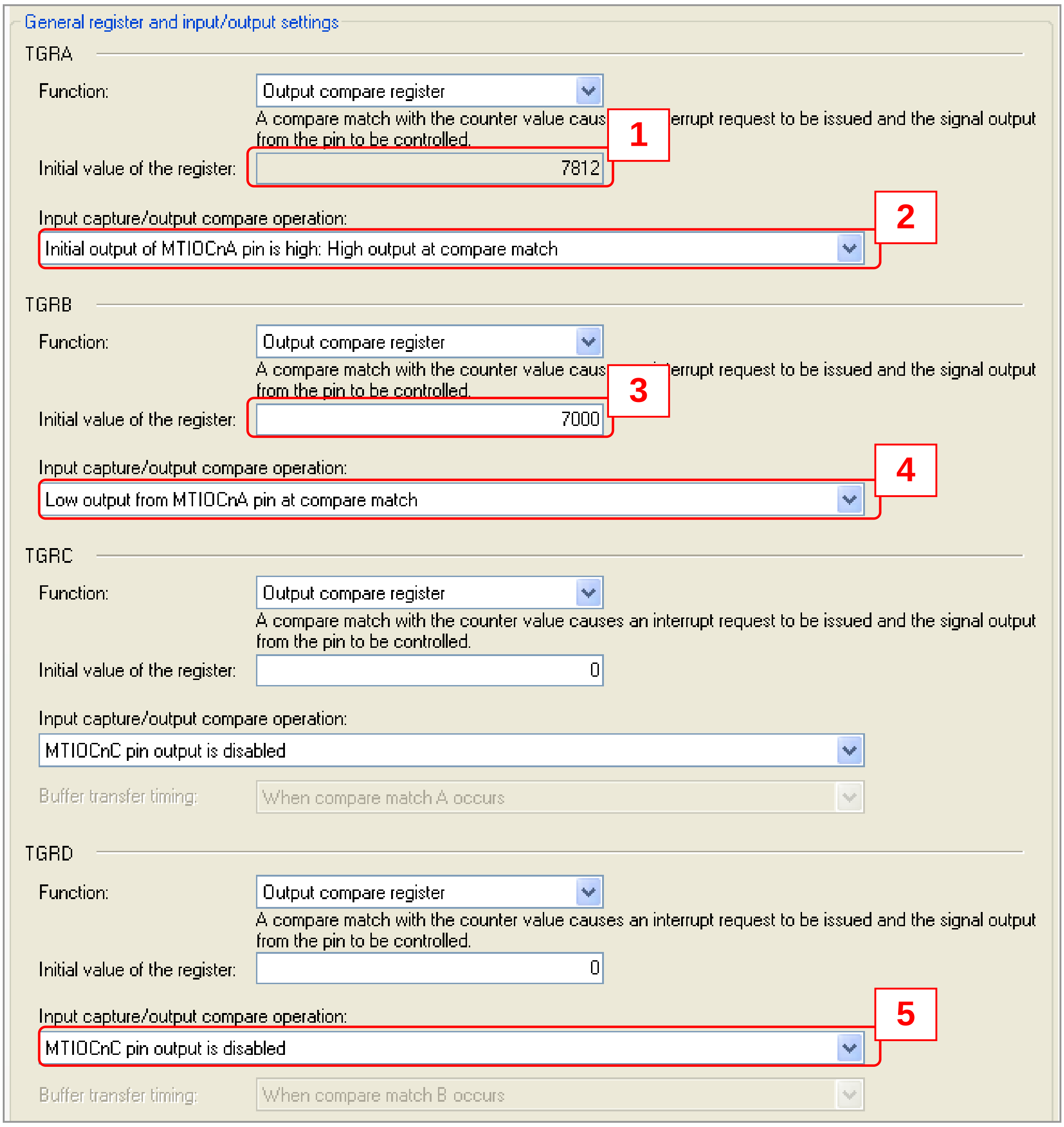
1. Select “TGRA compare match” for a counter clearing source.
2. Select “Internal clock (PCLK/256)” for a count source.
3. Set “500 msec” to timer operation period.



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

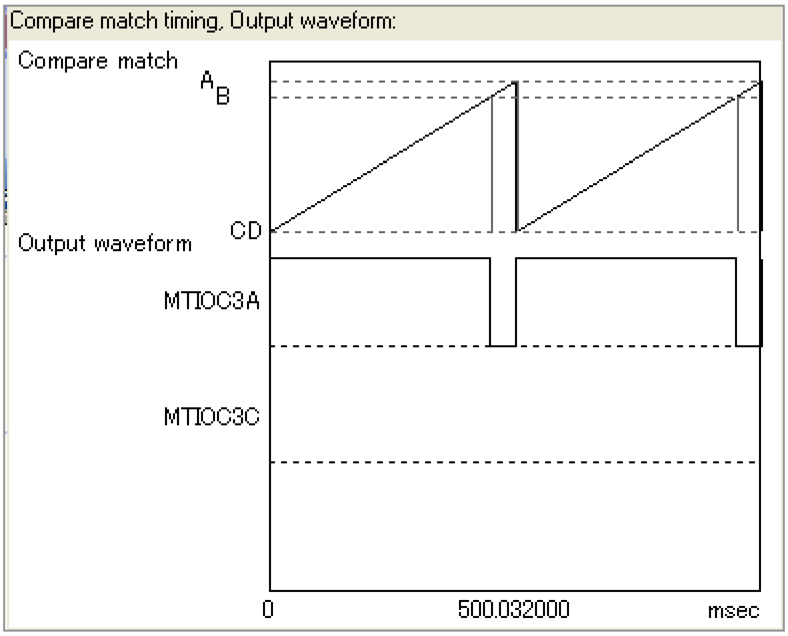
General register setting is as follows.

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 “7000” to TGRB initial value.
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.



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

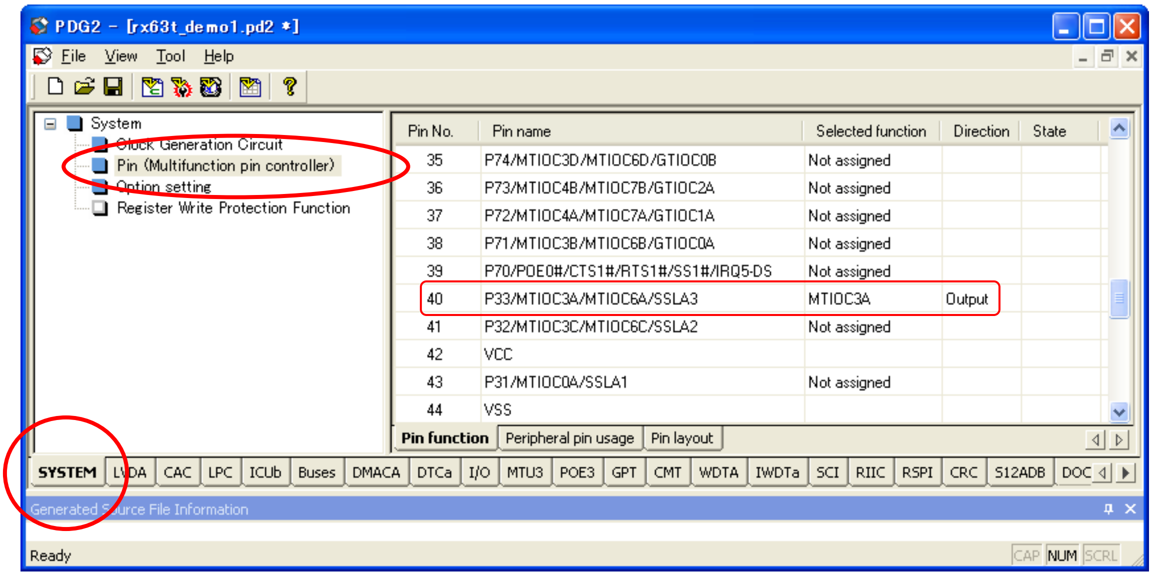
The compare match timing and the output waveform are displayed in a diagram.



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

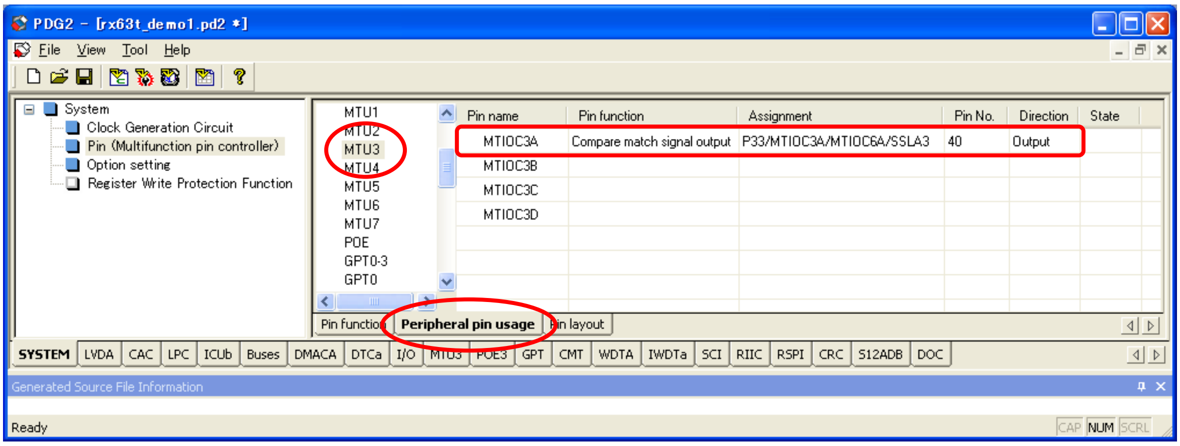
- It is possible to check the usage of pins on the pin function windows

1. After setting up the MTU3, select "SYSTEM" tab and click "Pin (Multifunction pin controller)" on the tree view.
2. On the Pin function window, you can see that No.40 pin is used as MTIOC3A.



- State of pin usage for each peripheral module is displayed in the peripheral pin usage window.

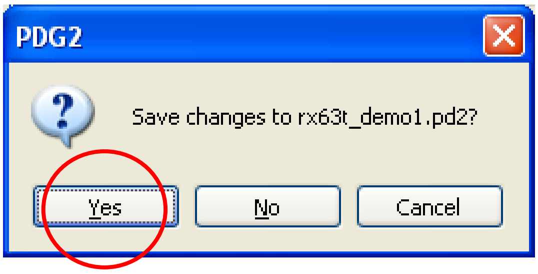
Select peripheral pin usage sheet and click MTU3 to check the usage of MTIOC3A pin.



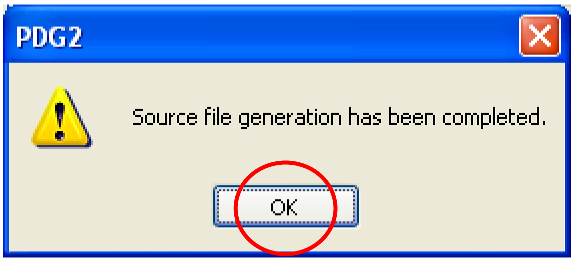
(11) Generating source files **PDG**

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

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

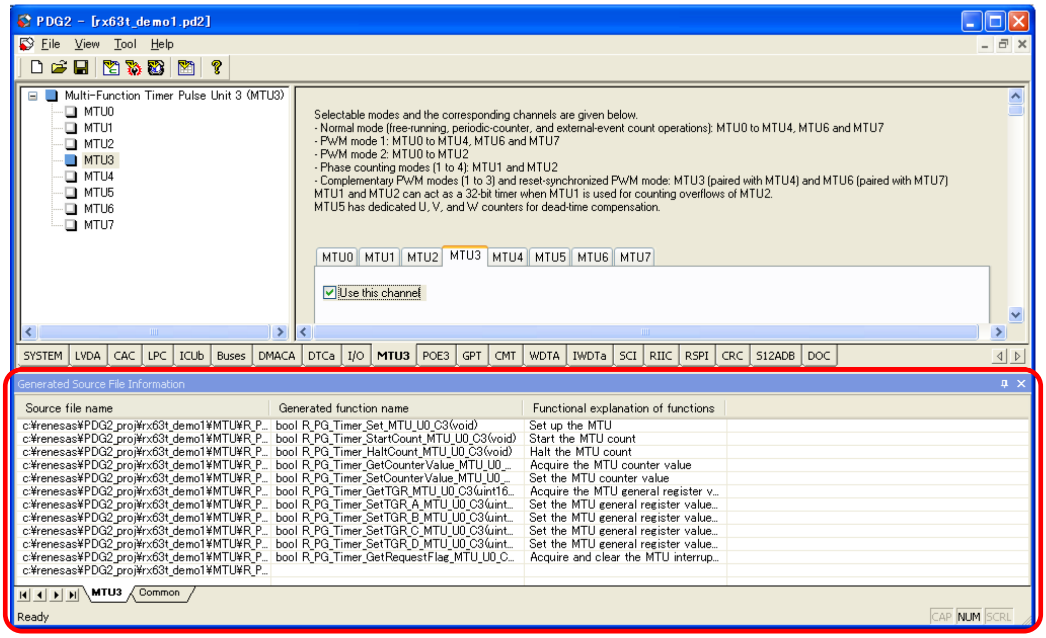


3. Click [OK] on the message box.



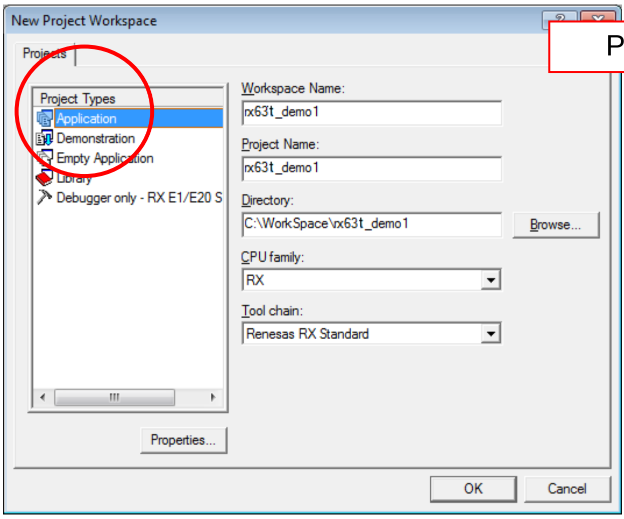
4. Generated functions are listed in lower pane.

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

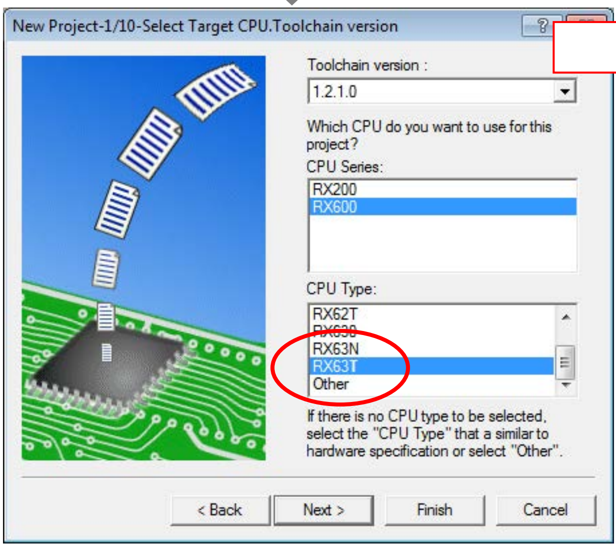


(12) Preparing the HEW project **HEW**

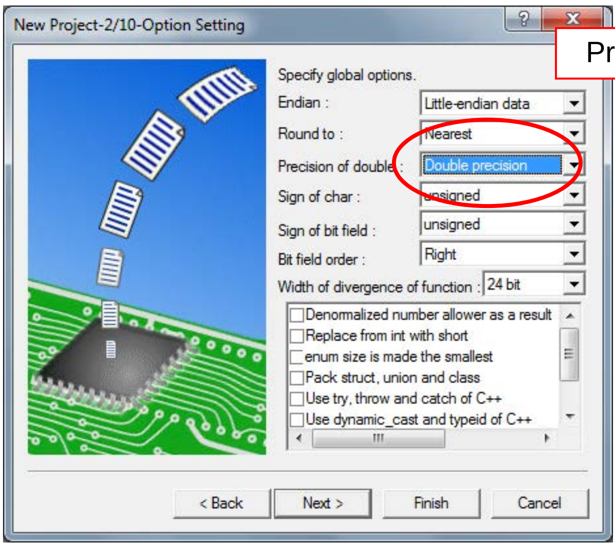
Start the HEW and make RX63T workspace.



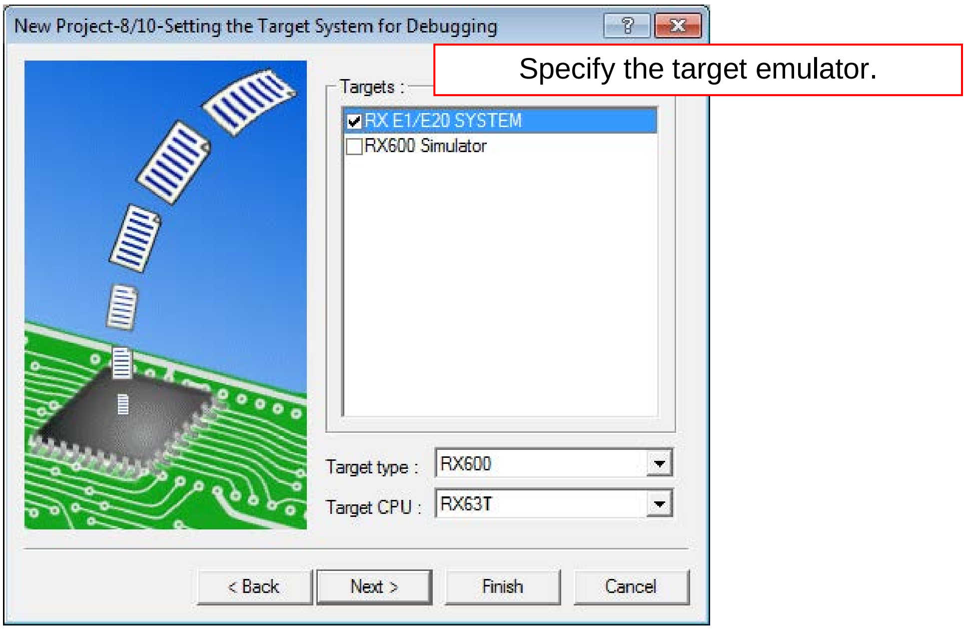
Project type : Application

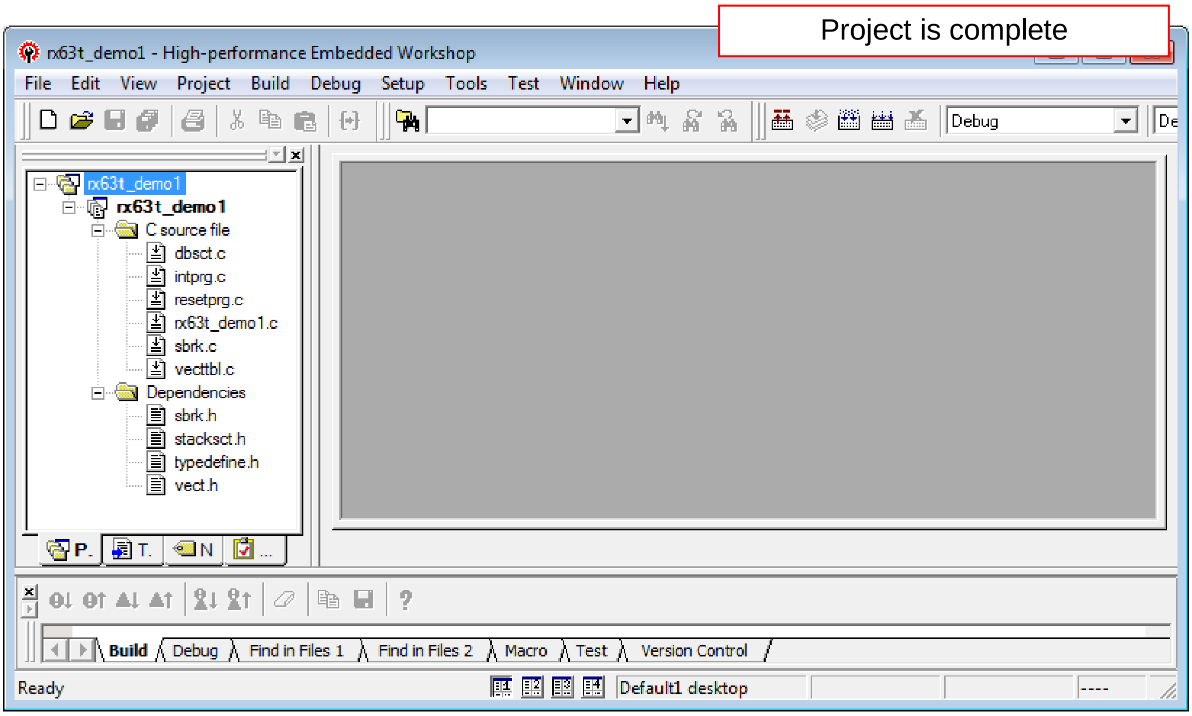


CPU type : RX63T



Precision of double : Double precision


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 : RX63T
< Back
Next >
Finish
Cancel

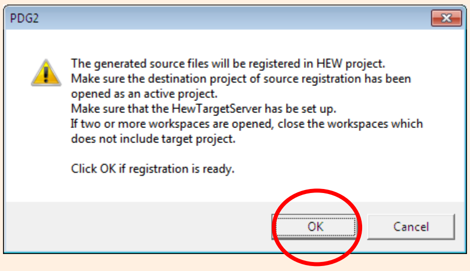

Project is complete
rx63t_demo1 - High-performance Embedded Workshop
File Edit View Project Build Debug Setup Tools Test Window Help
Debug
rx63t_demo1
rx63t_demo1
C source file
dbsct.c
intprg.c
resetprg.c
rx63t_demo1.c
sbrk.c
vecttbl.c
Dependencies
sbrk.h
stacksct.h
typedefine.h
vect.h
Build
Debug
Find in Files 1
Find in Files 2
Macro
Test
Version Control
Ready
Default1 desktop

(13) Adding the generated source files to the HEW project

PDG

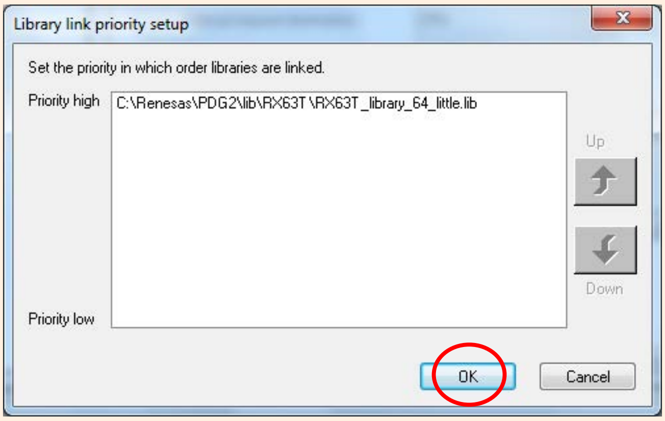
1. To add source files to HEW, click  on the tool bar.

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



3. This is a linkage setting of RPDL library.

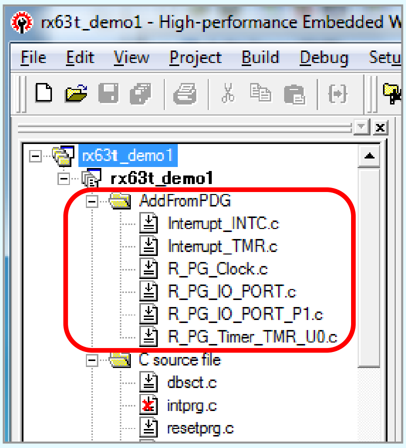
When using multiple lib files, linkage order can be set in this dialog box.



HEW

4. Source files are added to HEW

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 PDG user's manual.

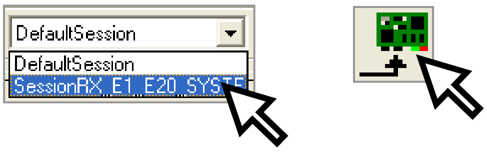
(14) Making the program on HEW **HEW**

By changing the part of “main” function, make the following program on HEW.

```
//Include "R_PG_<PDG project name>.h"
#include "R_PG_rx63t_demo1.h"

void main(void)
{
        //Configure I/O port pins that are not available
        R_PG_IO_PORT_SetPortNotAvailable();

        //Set up the clocks (wait cycle insertion)
        R_PG_Clock_WaitSet(0.01);

        //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);
}
```

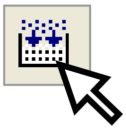
(15) Connecting to the emulator, building the program and executing **HEW**

1. Connect to the emulator



Connect button

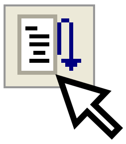
2. Just by clicking [Build] button, program can be built because RPDL library and include directory are automatically registered in build setting.



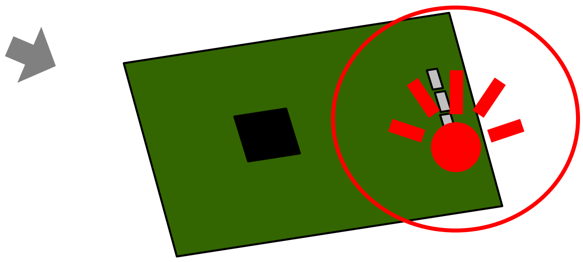
Build button

3. Download the program.

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

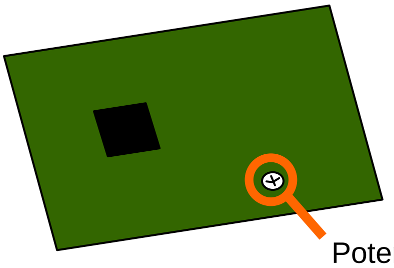


Reset go button



4.1.2 Continuously scanning on 12-Bit A/D converter (S12ADB)

In RX63T RSK board, the potentiometer is connected to AN000 analog input. In this tutorial, the 12-Bit A/D converter (S12ADB) will be set up to execute A/D conversion continuously. And the result of A/D conversion will be monitored on HEW.



Note : If there is a switch that enables/disables AN000 on the RSK board, enable it.

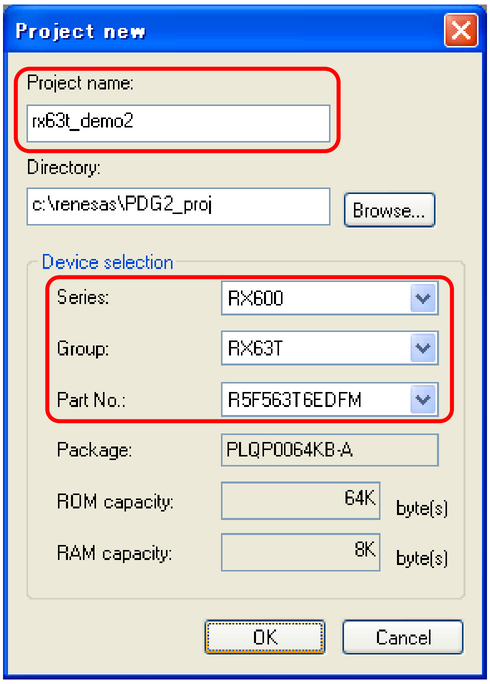
(1) Making the PDG project **PDG**

Make the new PDG project "rx63t_demo2". For details on how to make the new PDG project, refer to section 4.1.1 (1), Making the PDG project.

Set the CPU type as follows.

Series : RX600
Group : RX63T
Part No. : R5F563T6EDFM

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



(2) Clock setting **PDG**

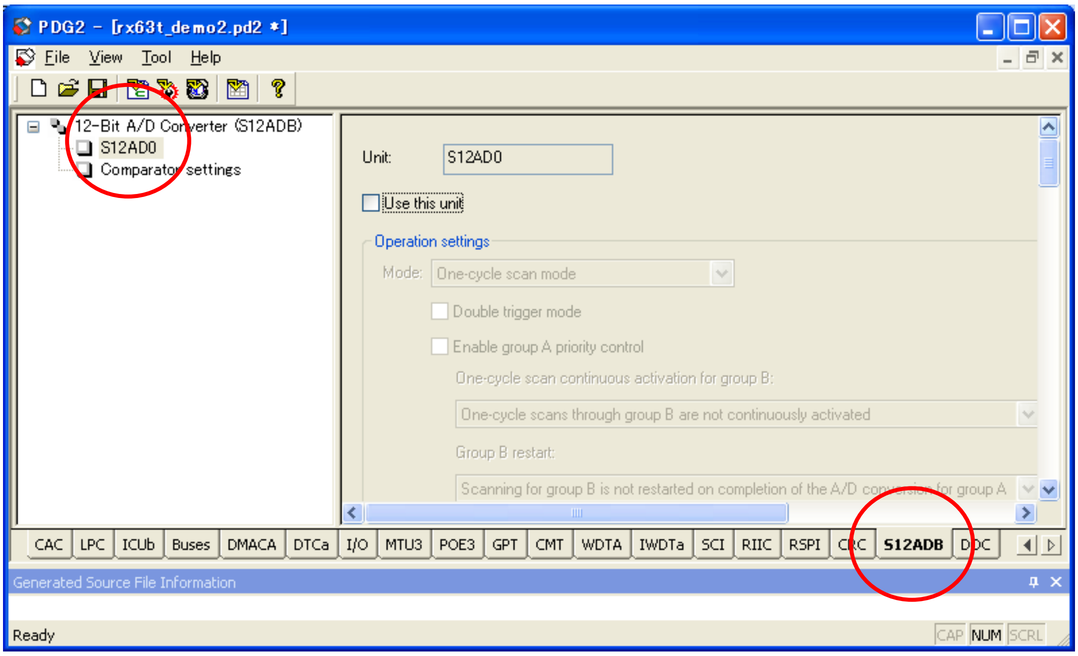
1. The clock setting window opens and the error icons are displayed in the initial state. For icons such as and displayed on window, refer to section 4.1.1 (2), Initial state.
2. For the clock setting, refer to section 4.1.1 (3), Clock setting.

(3) Endian setting **PDG**

For the endian setting, refer to section 3.3, Endian.

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

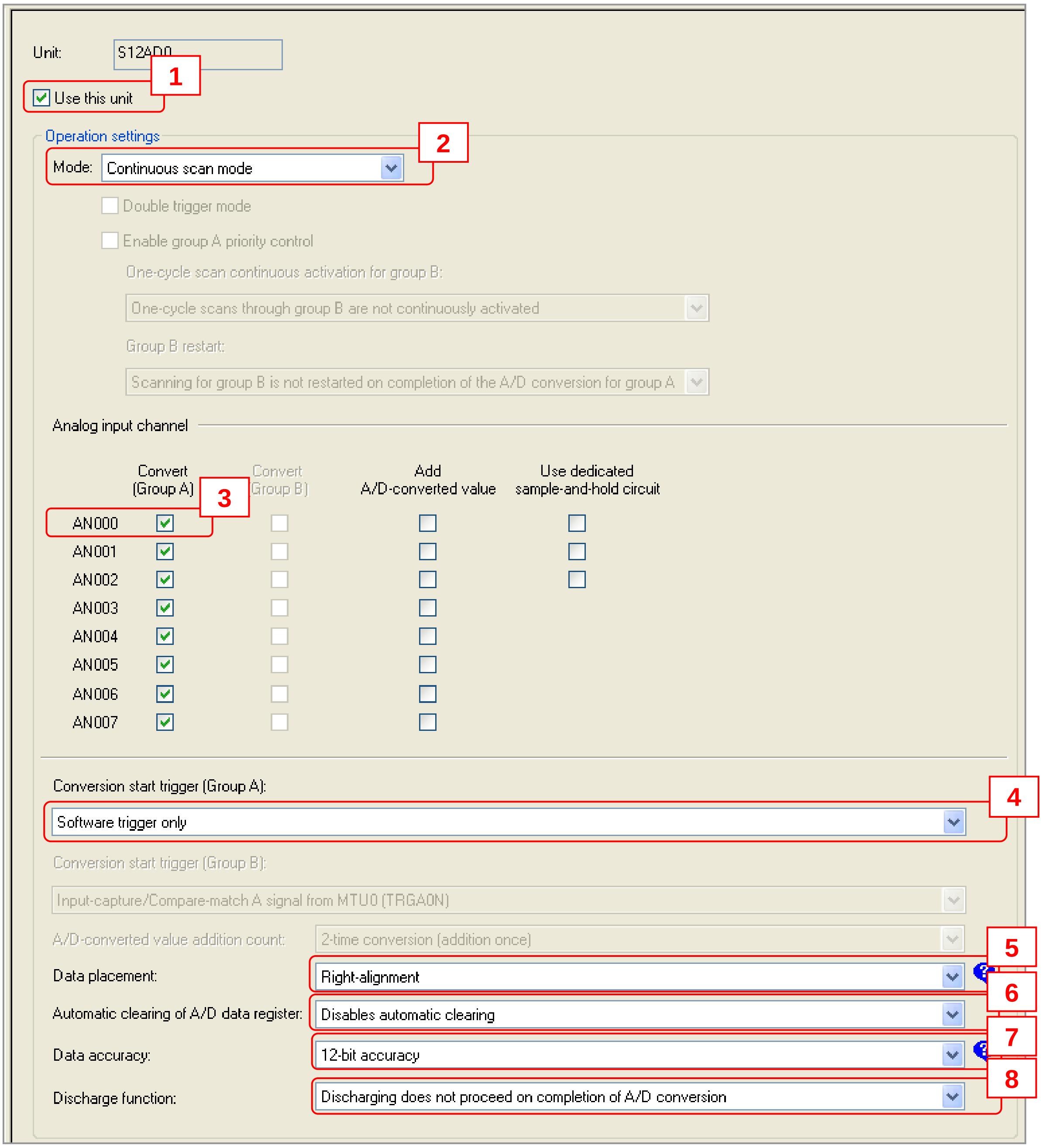
Select “S12ADB” tab and click S12AD0 on tree view.



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

Make the following setting for S12AD0.

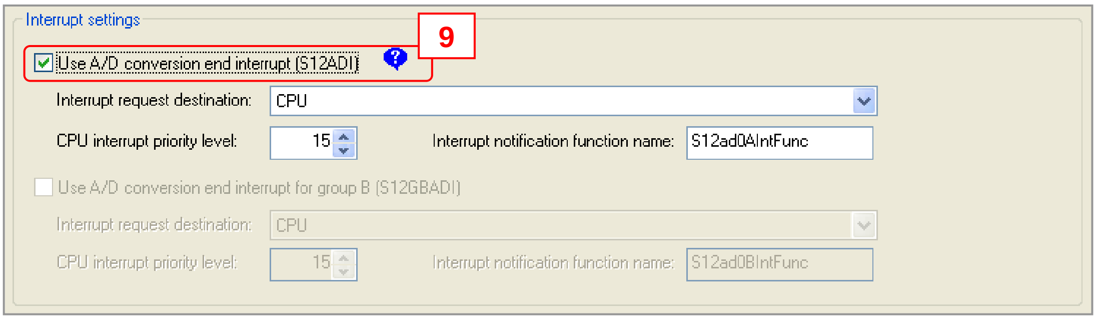
1. Check "Use this unit".
2. Select "Continuous scan mode" for the operation mode.
3. Check "AN000" for the analog input channel.
4. Select "Software trigger only" for the conversion start trigger.
5. Select "Right-alignment" for the data placement.
6. Select "Disables automatic clearing" for the automatic clearing of A/D data register.
7. Select"12-bit accuracy" for the data accuracy.
8. Select "Discharging does not proceed on completion of A/D conversion" for the discharge function.



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

Make the following setting for S12AD0.

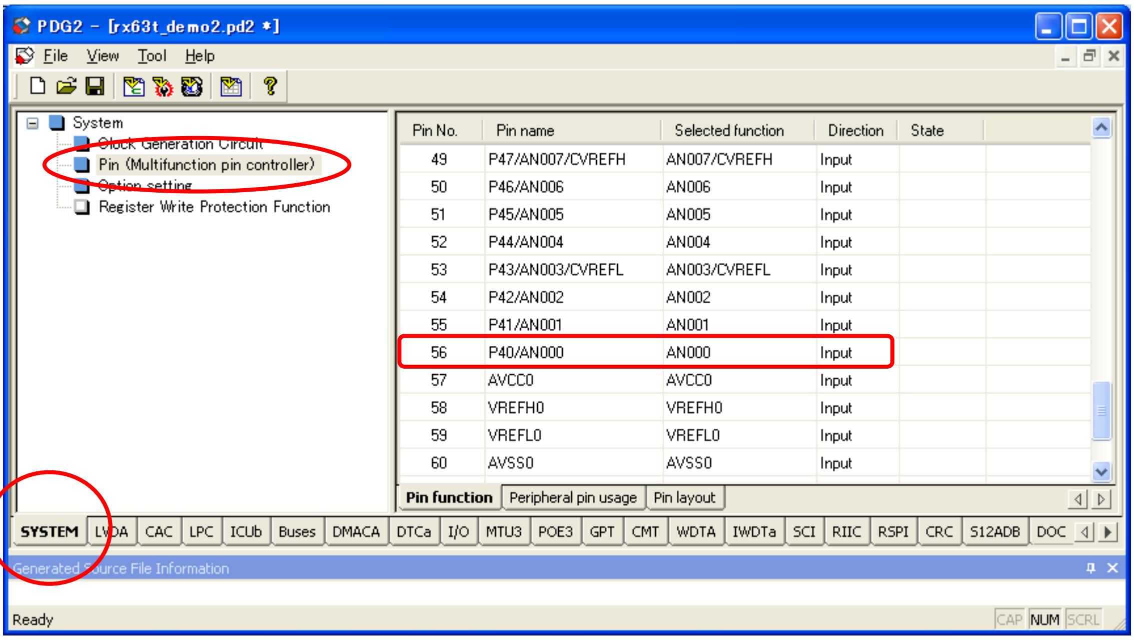
9. Check "Use A/D conversion end interrupt (S12ADI)".



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

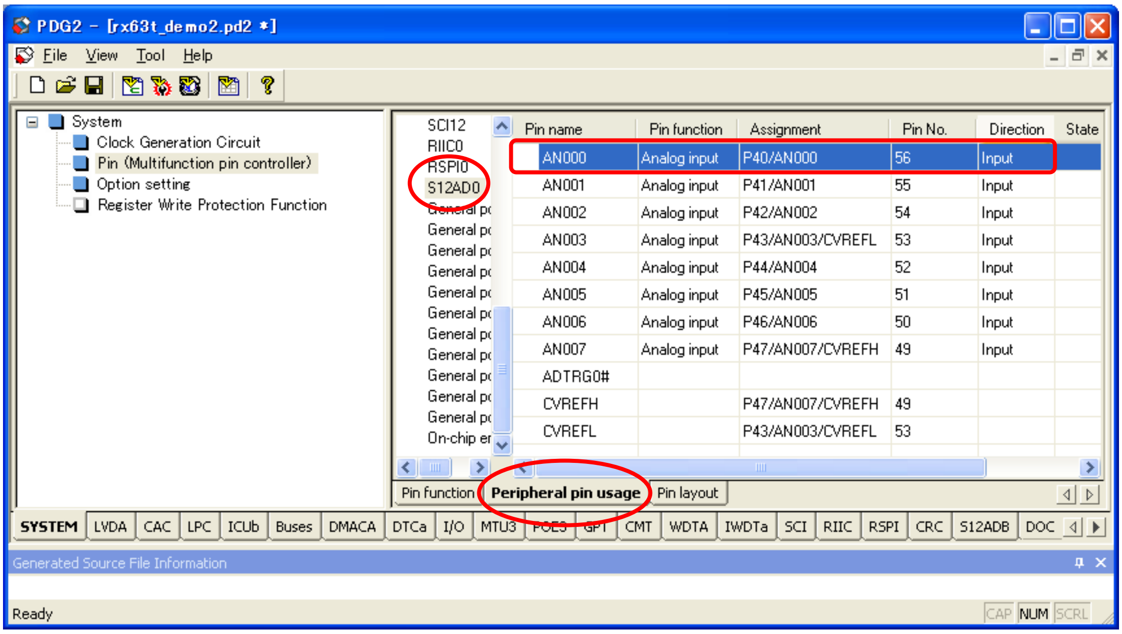
- It is possible to check the usage of pins on the pin function windows

1. After setting up the S12ADB, select "SYSTEM" tab and click "Pin" on the tree view.
2. On the Pin function window, you can see that No.56 pin is used as AN000.



- State of pin usage for each peripheral module is displayed in the peripheral pin usage window.

Select peripheral pin usage sheet and click S12AD0 to check the usage of AN000 pin.



(8) Generating source files **PDG**

To generate source files, click on the tool bar. For details on generating source files, refer to section 4.1.1 (9), Generating source files.

(9) Preparing the HEW project **HEW**

Start the HEW and make RX63T workspace. For details on making HEW project, refer to section 4.1.1 (10), Preparing the HEW project.

(10) Adding the generated source files to the HEW project **PDG**

To add the generated source files to HEW, click on the tool bar. For details on adding the source files to HEW project, refer to section 4.1.1 (11), Adding the generated source files to the HEW project.

(11) Making the program on HEW **HEW**

By changing the part of “main” function, make the following program on HEW.

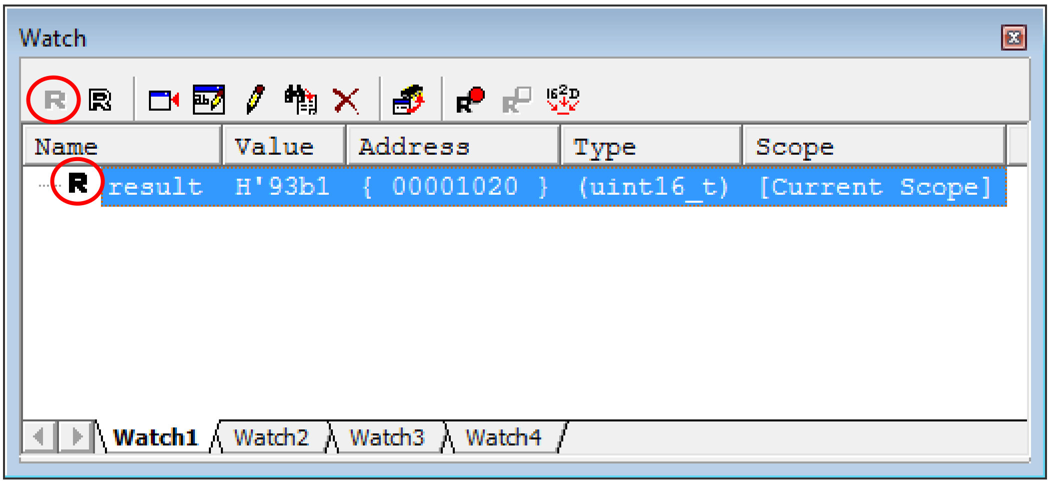
```
//Include "R_PG_<PDG project name>.h"
#include "R_PG_rx63t_demo2.h"

void main(void)
{
	//Set up the clocks (wait cycle insertion)
	R_PG_Clock_WaitSet(0.01);

	//Set up A/D converter
	R_PG_ADC_12_Set_S12AD0();

	//Start A/D conversion
	R_PG_ADC_12_StartConversion_S12AD0();

	while(1);
}

//Variable to store the result
uint16_t result;

//A/D conversion end interrupt notification function
void S12ad0AIntFunc(void)
{
	//Get the result of conversion
	R_PG_ADC_12_GetResult_S12AD0(&result, 0, 0, 0);
}
```

(12) 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 4.1.1 (13), connecting to the emulator, building the program and executing.

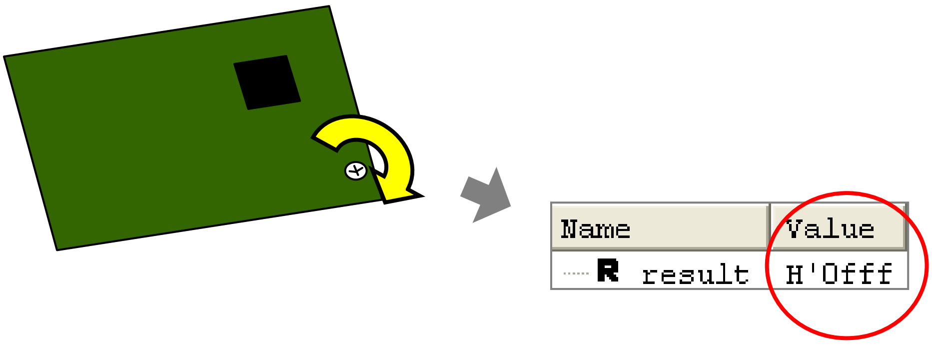
(13) Adding the variable of A/D conversion result to the watch window **HEW**

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



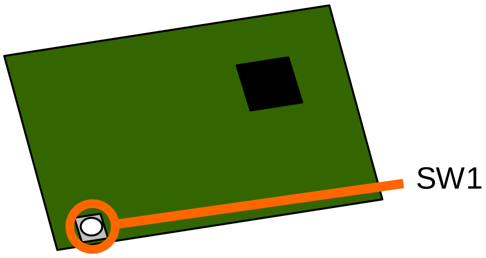
(14) Executing the program and monitoring the A/D conversion result **HEW**

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



4.1.3 Triggering DTCa by ICUb

In RX63T RSK board, switch 1 (SW1) is connected to IRQ0. In this tutorial, the data transfer controller (DTCa) and ICUb will be set up and DTC transfer triggered by IRQ0 will be performed.



Note : If there is a switch that enables/disables IRQ0 on the RSK board, enable it.

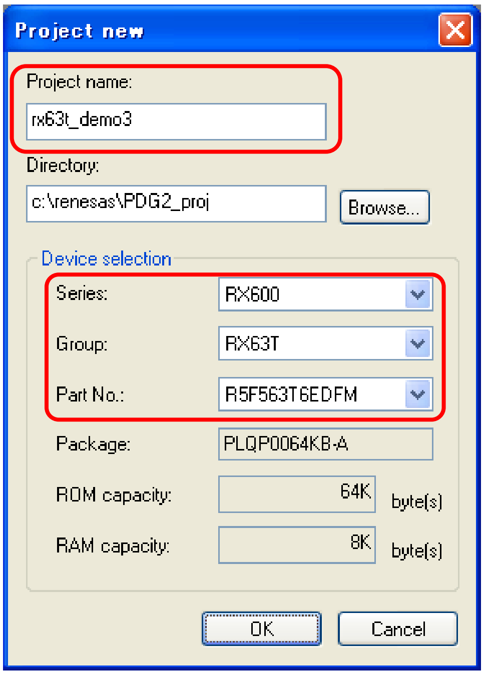
(1) Making the PDG project **PDG**

Make the new PDG project “rx63t_demo3”. For details on how to make the new PDG project, refer to section 4.1.1 (1), Making the PDG project.

Set the CPU type as follows.

Series : RX600
Group : RX63T
Part No. : R5F563T6EDFM

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



(2) Clock setting **PDG**

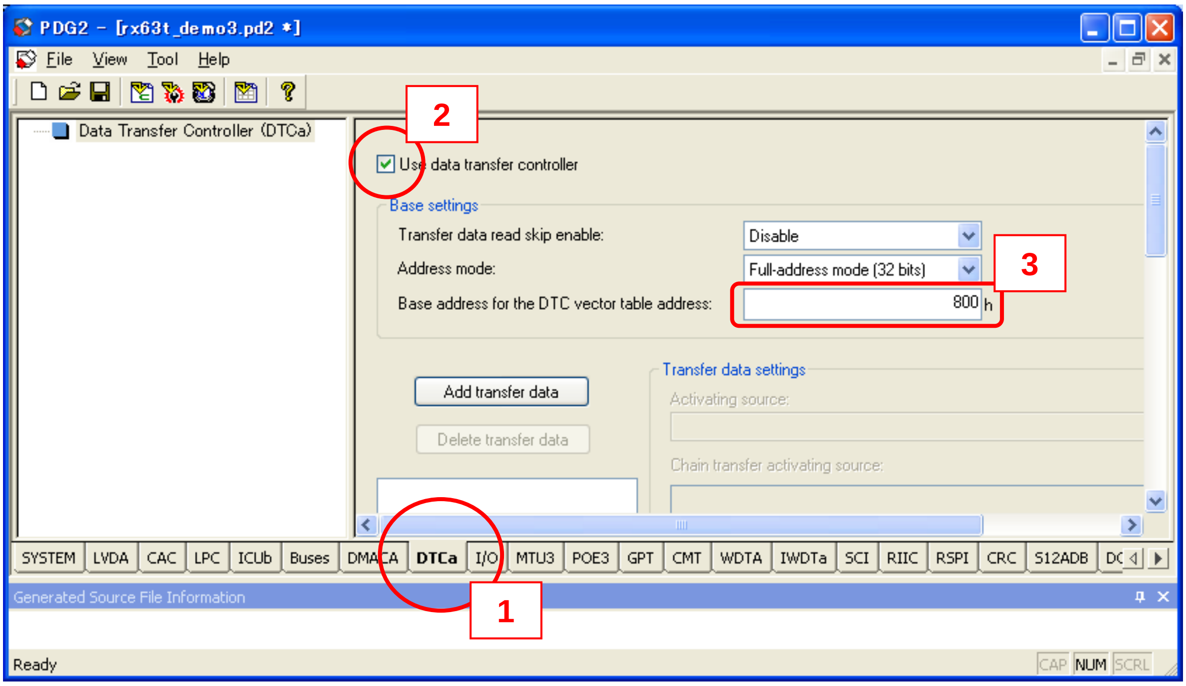
1. The clock setting window opens and the error icons are displayed in the initial state. For icons such as and displayed on window, refer to section 4.1.1 (2), Initial state.
2. For the clock setting, refer to section 4.1.1 (3), Clock setting.

(3) Endian setting **PDG**

For the endian setting, refer to section 3.3, Endian.

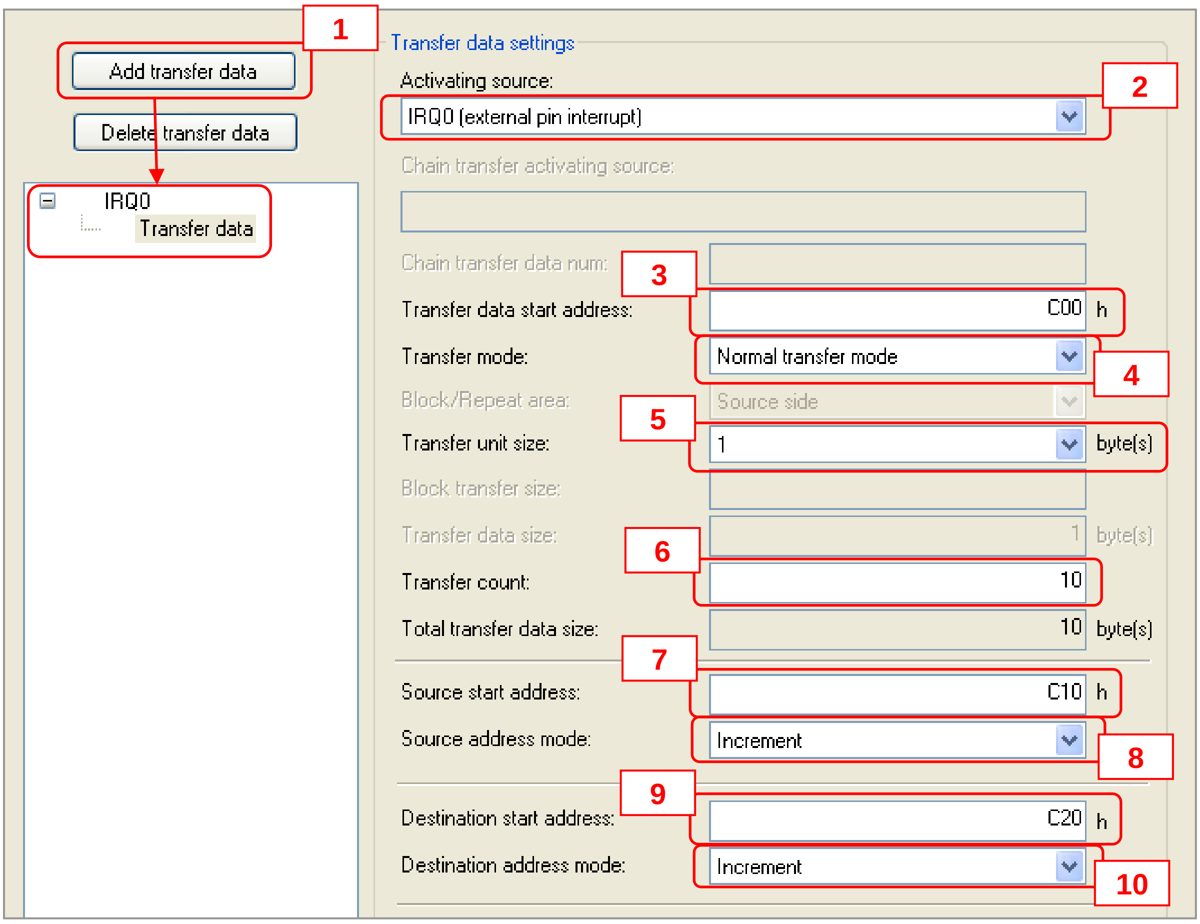
(4) DTCa setting-1 **PDG**

1. Select "DTCa" tab to open the DTCa setting window.
2. Check "Use data transfer controller".
3. The DTCa vector table will be allocated from 800h. Set "800".



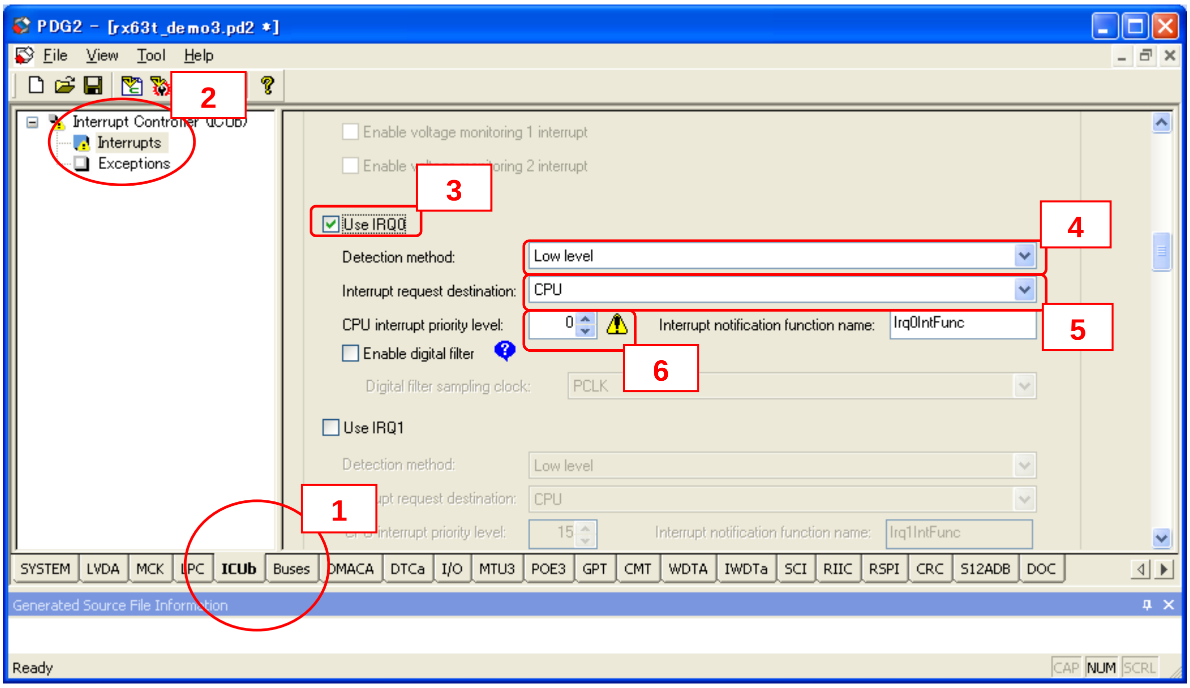
(5) DTCa setting-2 **PDG**

1. Click [Add transfer data] to add the transfer data.
2. Select “IRQ0 (external pin interrupt)” for the activating source.
3. Set “C00” to the transfer data start address.
4. Select “Normal transfer mode” for the transfer mode.
5. Set “1” to the transfer unit size.
6. Set “10” to the transfer count.
7. Set “C10” to the source start address.
8. Select “Increment” for the source address mode.
9. Set “C20” to the destination start address.
10. Select “Increment” for the destination address mode.



(6) ICUb setting **PDG**

1. Select “ICUb” tab to open the ICUb setting window.
2. Click “Interrupts” on the tree view.
3. Check "Use IRQ0".
4. Select “Falling edge” for the detection method of IRQ0.
5. Select “CPU (After activating DTC and data transfer completion)”.
6. CPU interrupt will not be used then set “0” to the CPU interrupt priority level.



(7) Generating source files **PDG**

To generate source files, click  on the tool bar. For details on generating source files, refer to section 4.1.1 (9), Generating source files.

(8) Preparing the HEW project **HEW**

Start the HEW and make RX63N workspace. For details on making HEW project, refer to section 4.1.1 (10), Preparing the HEW project.

(9) Adding the generated source files to the HEW project **PDG**

To add the generated source files to HEW, click  on the tool bar. For details on adding the source files to HEW project, refer to section 4.1.1 (11), Adding the generated source files to the HEW project.

(10) Making the program on HEW **HEW**

By changing the part of “main” function, make the following program on HEW.

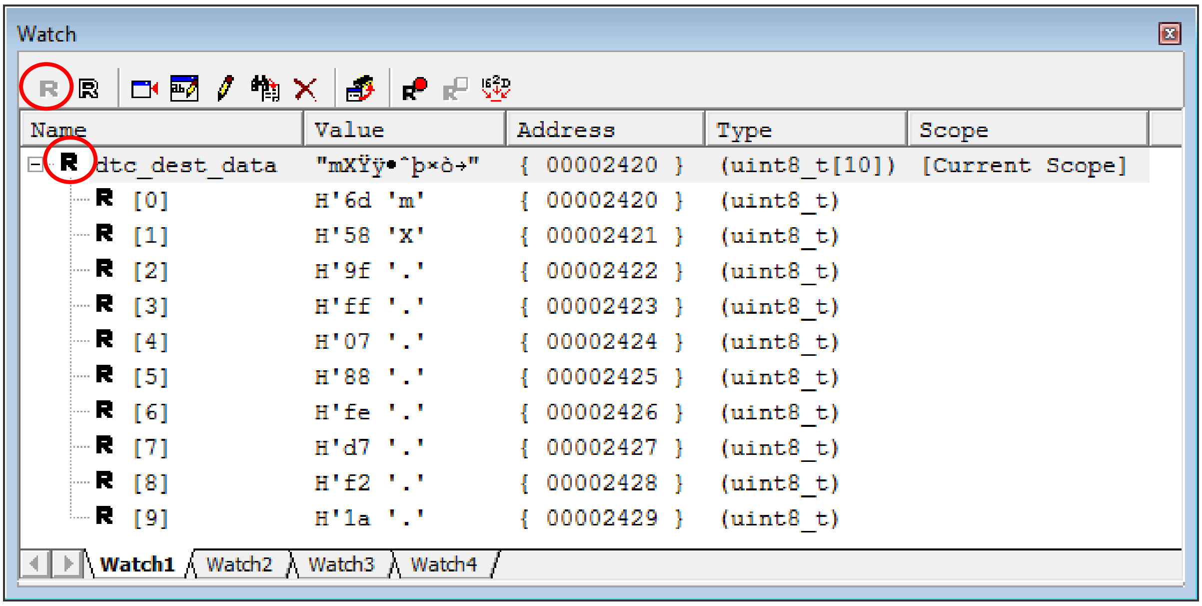
```
//Include "R_PG_<PDG project name>.h"
#include "R_PG_rx63t_demo3.h"

//DTC vector table
#pragma address dtc_vector_table = 0x00000800
uint32_t dtc_vector_table [256];

//DTC transfer data storage area (IRQ0)
#pragma address dtc_transfer_data_IRQ0 = 0x00000C00
uint32_t dtc_transfer_data_IRQ0 [4];

//Transfer source
#pragma address dtc_src_data = 0x00000C10
uint8_t dtc_src_data [10] = "ABCDEFGHIJ";

//Transfer destination
#pragma address dtc_dest_data = 0x00000C20
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;
	}

	R_PG_Clock_WaitSet(0.01);	//Set up the clocks (wait cycle insertion)

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

	// Set up the DTC (transfer data of IRQ0)
	R_PG_DTC_Set_IRQ0();

	R_PG_ExtInterrupt_Set_IRQ0();	// Set up IRQ0

	R_PG_DTC_Activate();	// Make the DTC be ready to the trigger

	while(1);
}
```

(11) 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 4.1.1 (13), connecting to the emulator, building the program and executing.

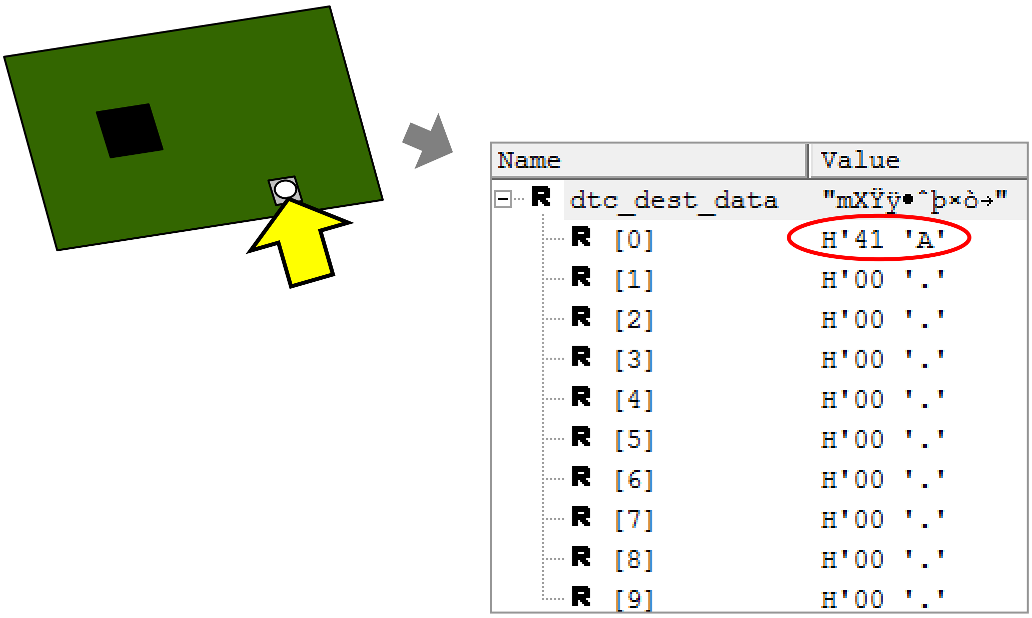
(12) Adding the variable of the transfer destination **HEW**

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.



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

Start the execution and push the SW1. The value of “dtc_dest_data” on the watch window will change.



4.2 When the CubeSuite+ is in Use with Renesas starter kit for RX63T(64-pin)

This section introduces the usage of the PDG by giving instructions on how to use the PDG and CubeSuite+ to create a tutorial program that implements the following operations on the Renesas Starter Kit board for the RX63T(64-pin).

・ An LED blinking on Compare Match Timer (CMT) interrupt

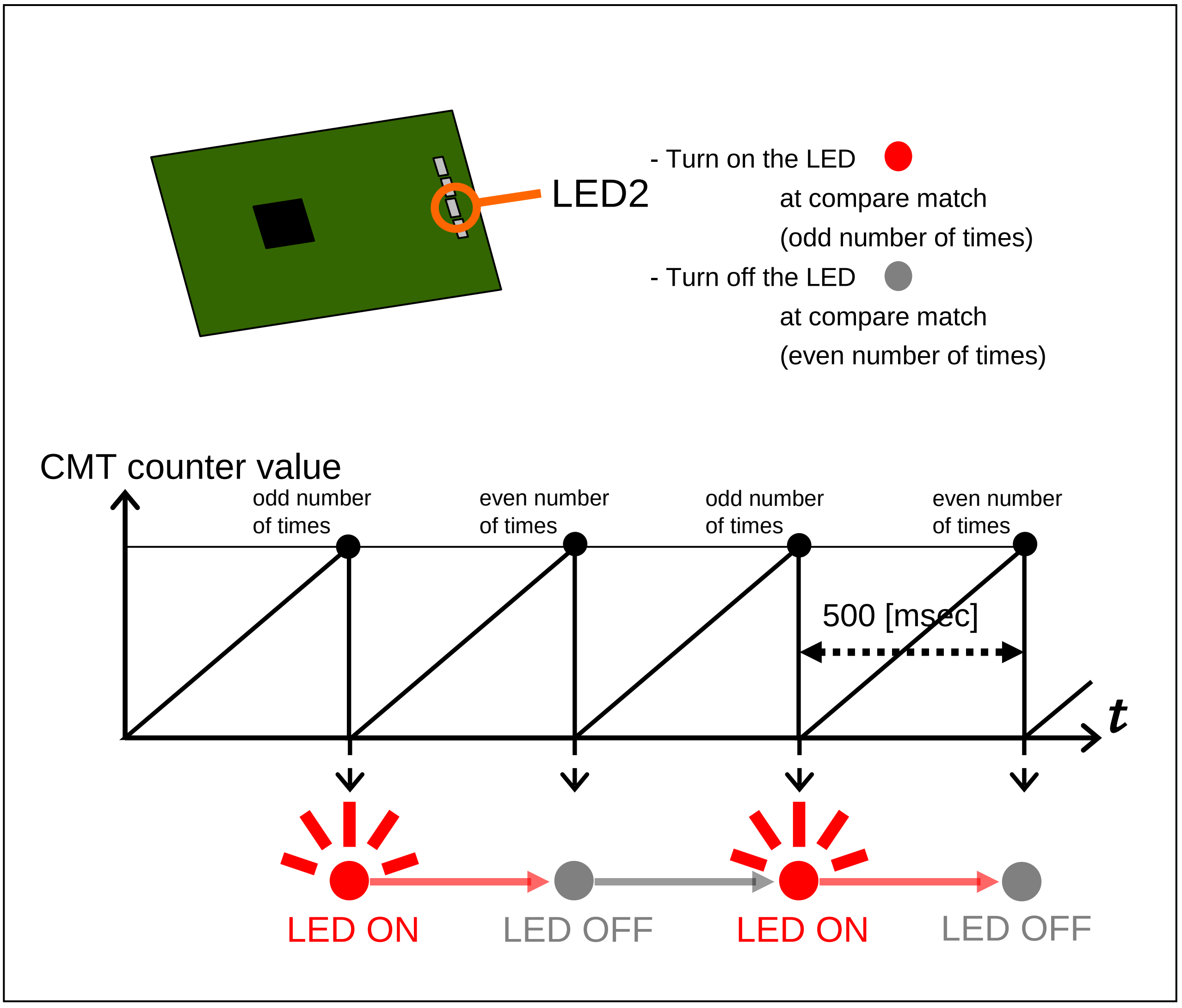
The labels given below respectively indicate operations to take place in the PDG and in the CubeSuite+.

PDG	: Operations in the PDG
CubeSuite+	: Operations in the CubeSuite+

4.2.1 An LED blinking on Compare Match Timer (CMT) interrupt

The LED2 on RSK board is connected to P73. In this tutorial, Compare Match Timer and I/O port will be set up to blink this LED as follows.

Note : If there is a switch that enables/disables P73 on the RSK board, enable it.



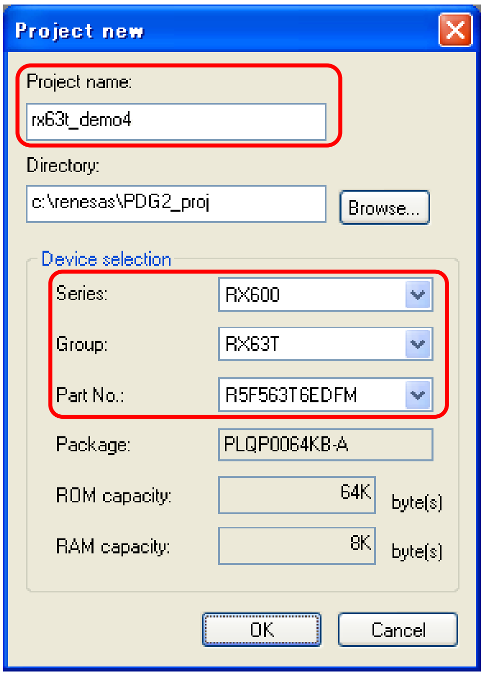
(1) Making the PDG project **PDG**

Make the new PDG project "rx63t_demo5". For details on how to make the new PDG project, refer to section 4.1.1 (1), Making the PDG project.

Set the CPU type as follows.

Series : RX600
Group : RX63T
Part No. : R5F563T6EDFM

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 4.1.1 (2), Initial state.
2. For the clock setting, refer to section 4.1.1 (3), Clock setting.

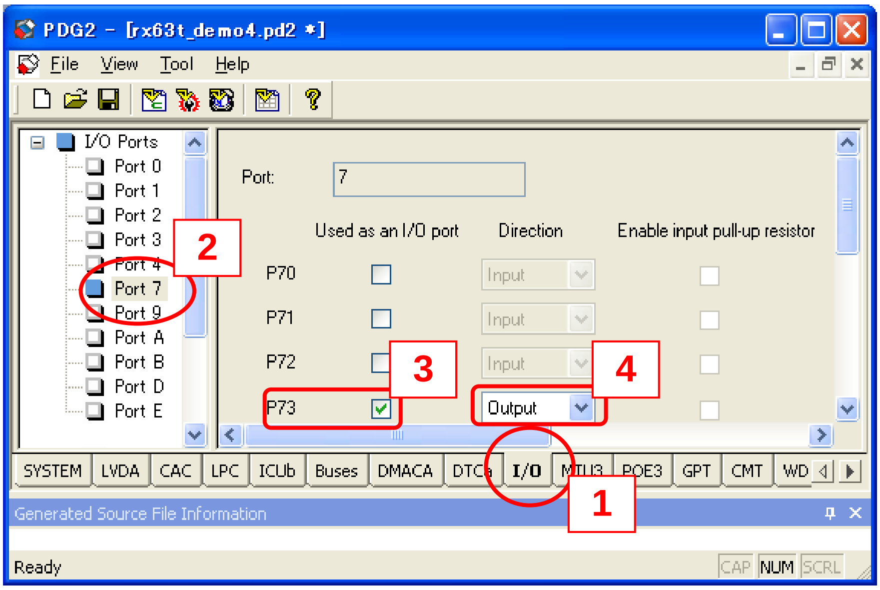
(3) Endian setting **PDG**

For the endian setting, refer to section 3.3, Endian.

(4) I/O Port setting PDG

The LED2 on RSK is connected to P so set P73 to output port.

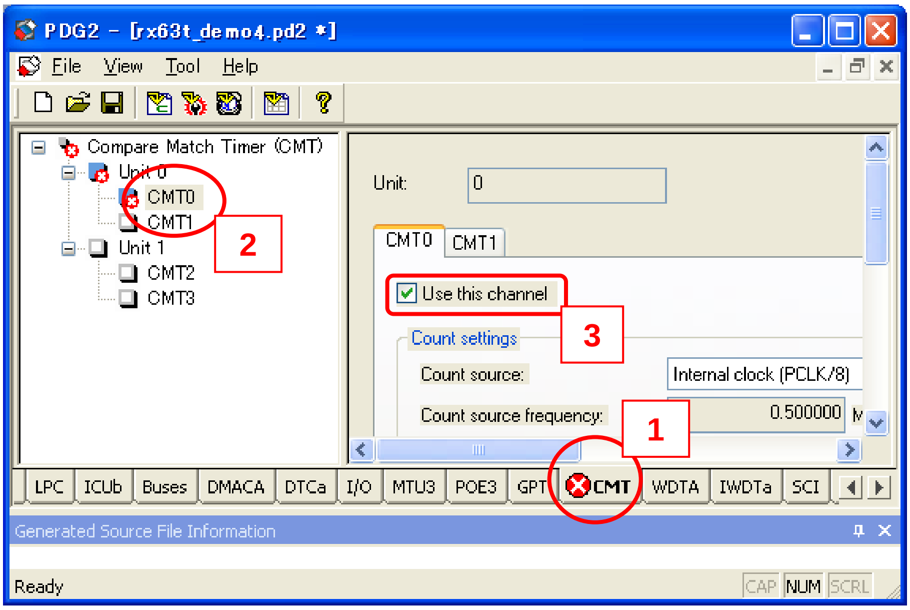
1. Select “I/O” tab
2. Select “Port 7”
3. Check “P73”
4. Select “Output”



(5) CMT setting-1 PDG

In this tutorial, CMT0 is used.

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

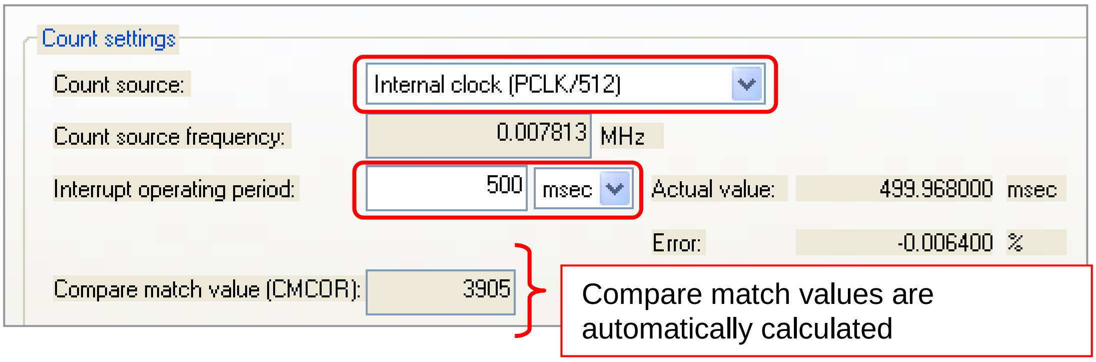


(6) CMT setting-2 **PDG**

Set the other items as follows.

-Count source: Internal clock(PCLK/512)

-Interrupt operating period: 500 msec



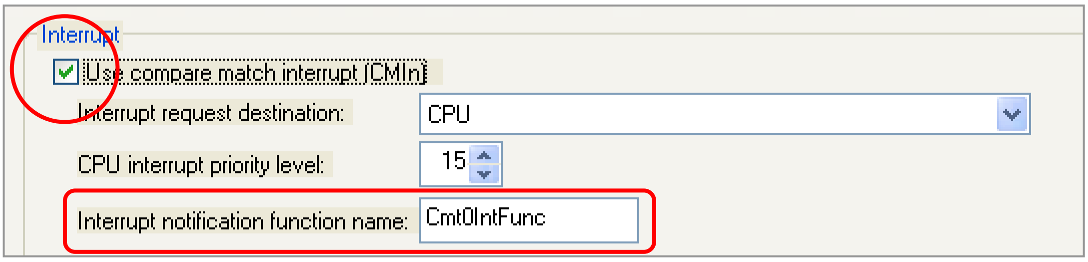
(7) CMT setting-3 **PDG**

Set the interrupt notification functions.

This functions are called when the interrupt occurs.

-Check “Use compare match interrupt (CMIn)”

Notification function name is "Cmt0IntFunc"



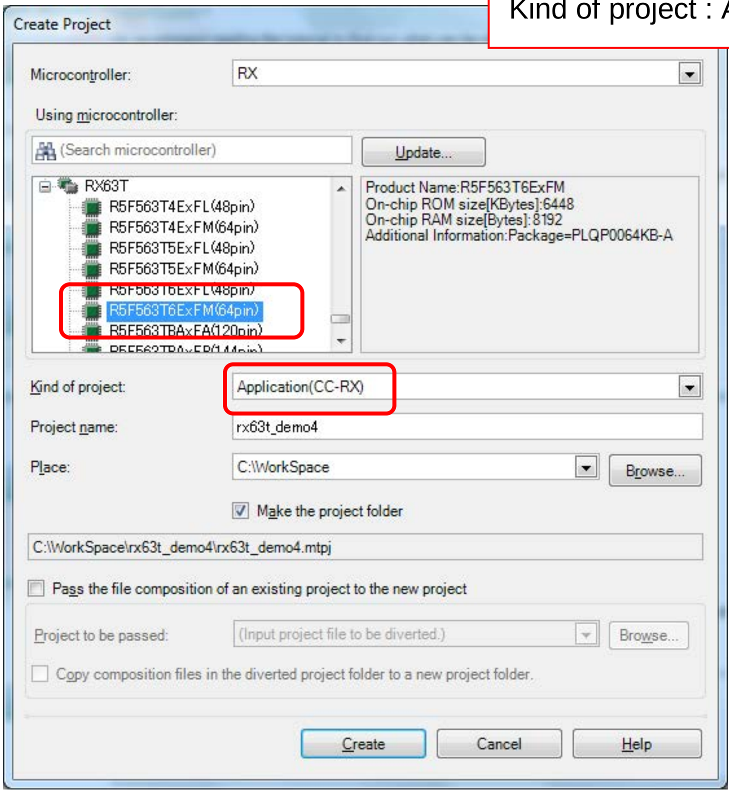
(8) Generating source files **PDG**

To generate source files, click  on the tool bar. For details on generating source files, refer to section 4.1.1 (9), Generating source files.

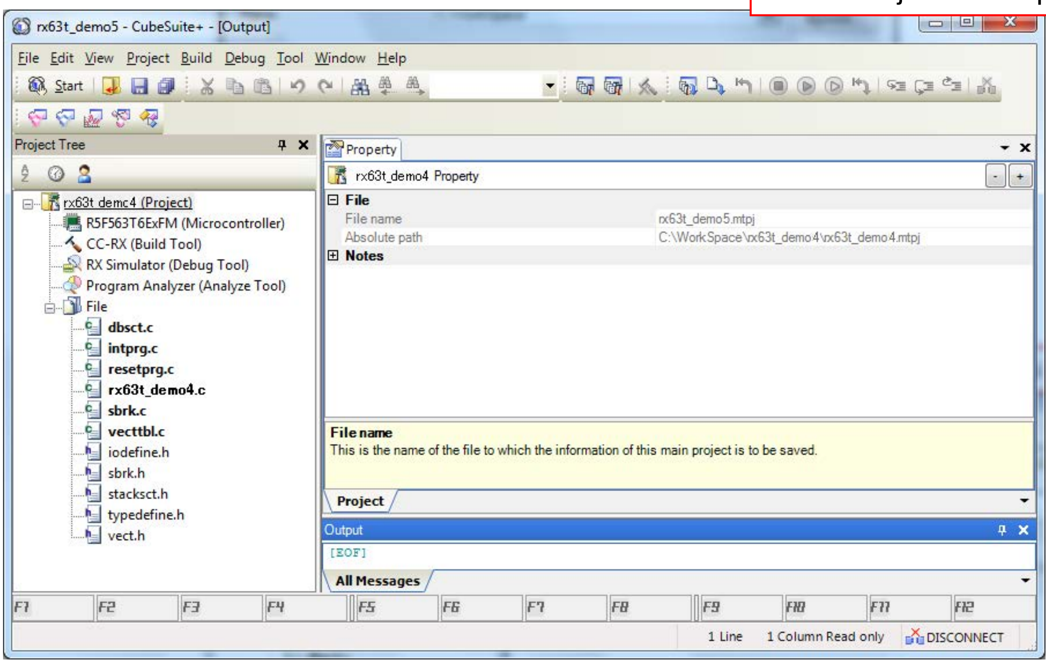
(9) Preparing the CubeSuite+ project **CubeSuite+**

Start the CubeSuite+ and make RX63T workspace.

Using microcontroller : R5F563T6ExFM(64pin)
Kind of project : Application(CC-RX)



Project is complete



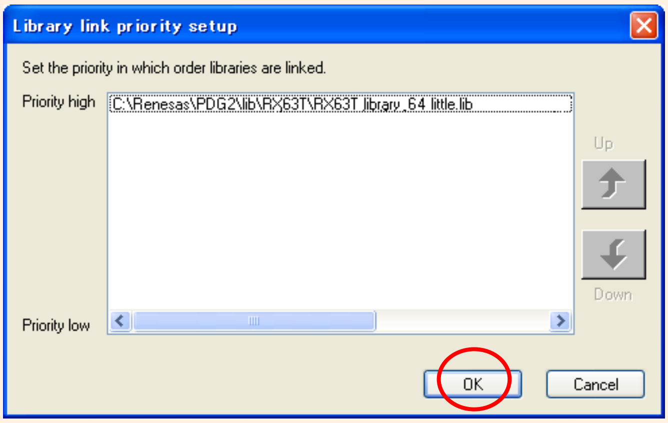
(10) Adding the generated source files to the CubeSuite+ project

PDG

1. To add source files to CubeSuite+, click  on the tool bar.

2. This is a linkage setting of RPDL library.

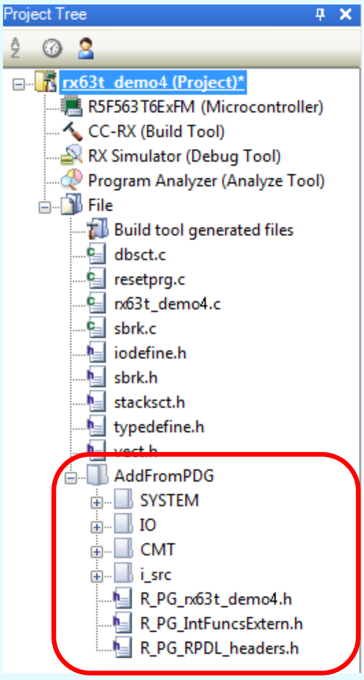
   When using multiple lib files, linkage order can be set in this dialog box.



CubeSuite+

3. Source files are added to CubeSuite+.

   Added source files are put in "AddFromPDG" category.



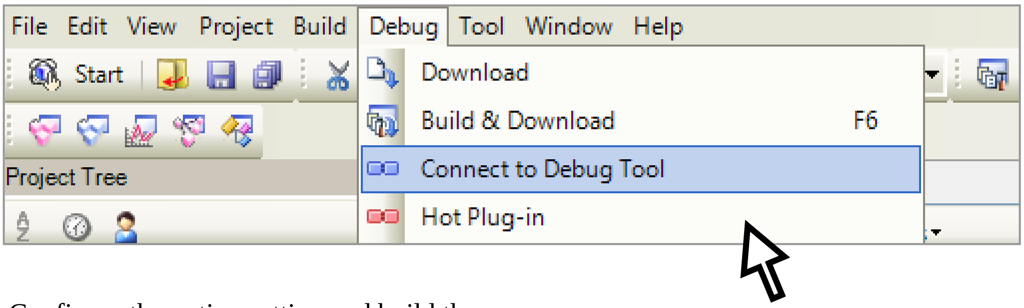
(11) Making the program on CubeSuite+ **CubeSuite+**

By changing the part of “main” function, make the following program on CubeSuite+.

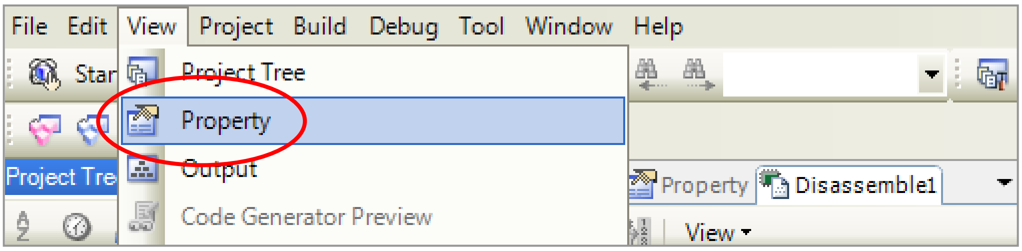
```
//Include "R_PG_<PDG project name>.h"
#include "R_PG_rx63t_demo4.h"

bool led=false;

void main(void)
{
	//Set up the clocks (wait cycle insertion)
	R_PG_Clock_WaitSet(0.01);

	//Set up port P73
	R_PG_IO_PORT_Write_P73(1);	//Initial output value
	R_PG_IO_PORT_Set_P73();

	//Set up the CMT
	R_PG_Timer_Set_CMT_U0_C0();

	//Start the CMT count
	R_PG_Timer_StartCount_CMT_U0_C0();

	while(1);
}

// Compare match interrupt notification function
void Cmt0IntFunc(void)
{
	if( led ){
		//Turn off the LED
		R_PG_IO_PORT_Write_P73(1);
		led = false;
	}
	else{
		//Turn on the LED
		R_PG_IO_PORT_Write_P73(0);
		led = true;
	}
}
```

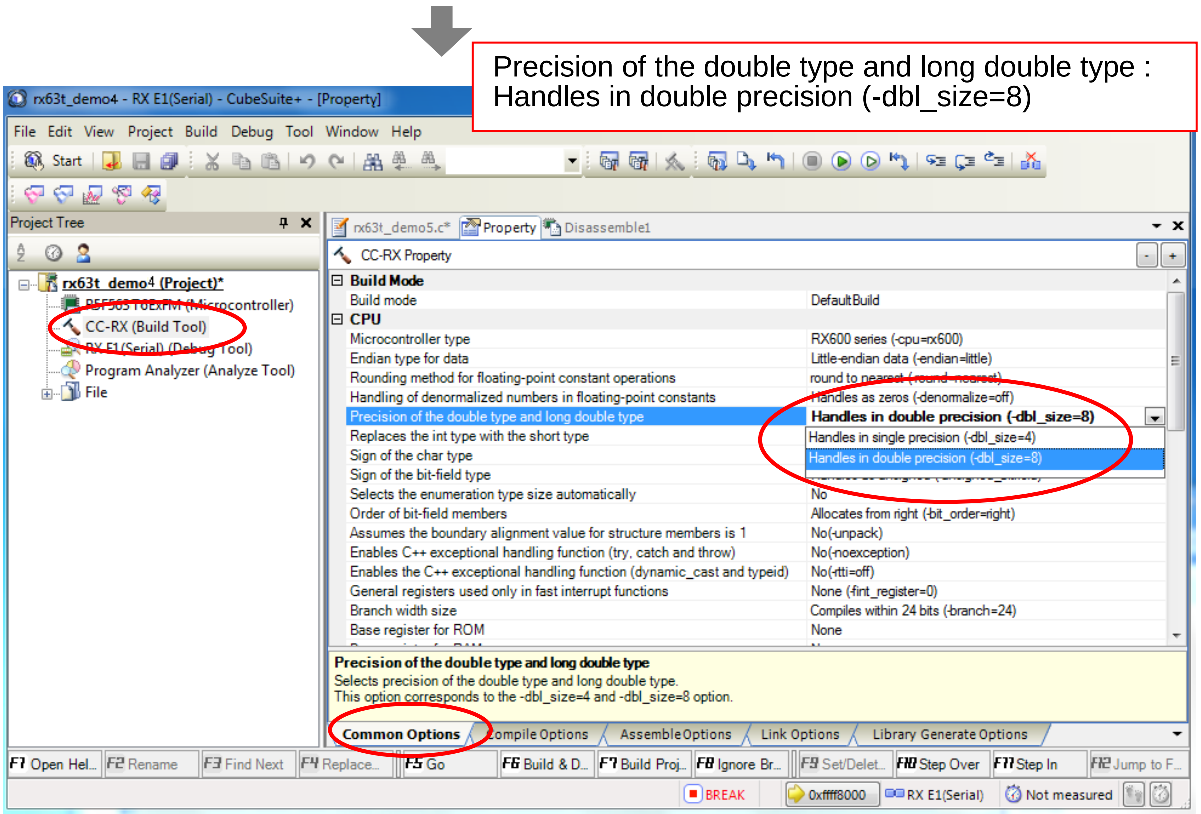
(12) Connecting to the emulator, building the program, downloading and executing **CubeSuite+**

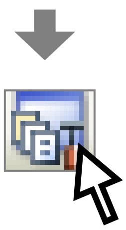
1. Connect to the emulator.



2. Configure the option setting and build the program.

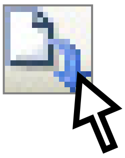






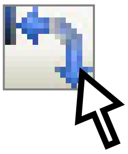
Build button

3. Download the program.

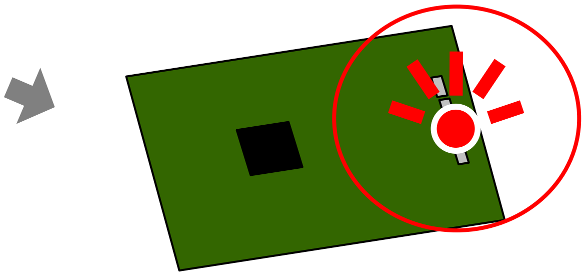


Download button

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



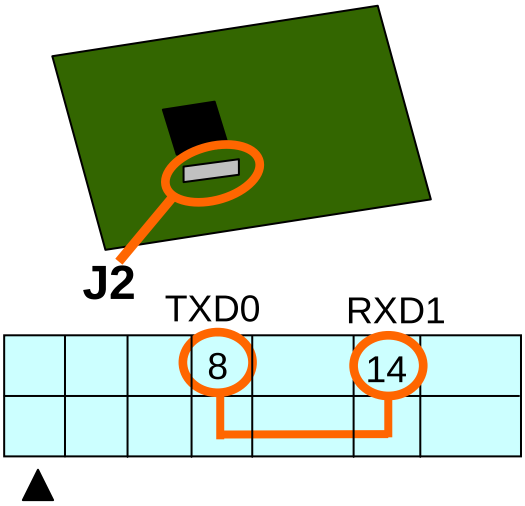
Reset go button



4.3 When the e2 studio is in Use with Renesas starter kit for RX63T(64-pin)

4.3.1 Data transfer between SCIc channels 0 and 1

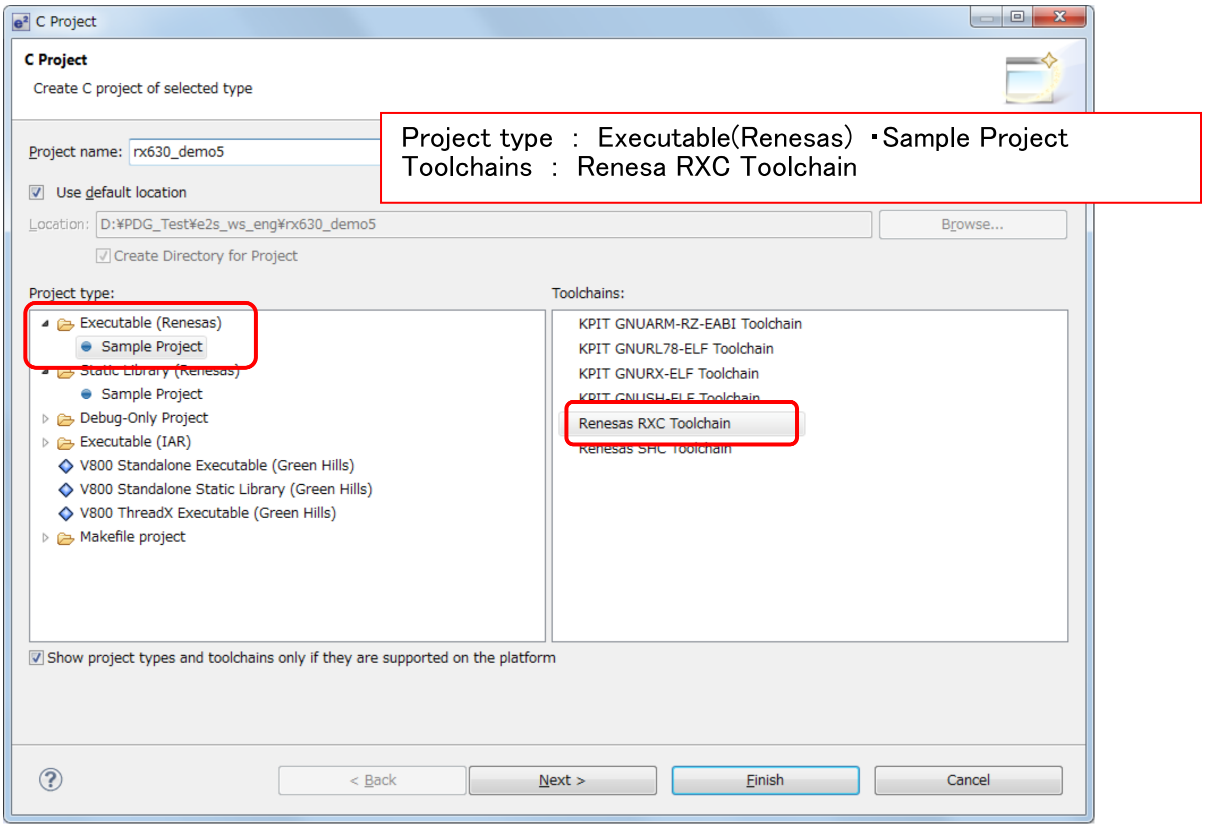
In this tutorial, SCI channel 0 and 1 will be set up to transfer data in asynchronous mode. Connect the transmission pin of channel 0 (TXD0) and the reception pin of channel 1 (RXD1) on the RSK board as follows.

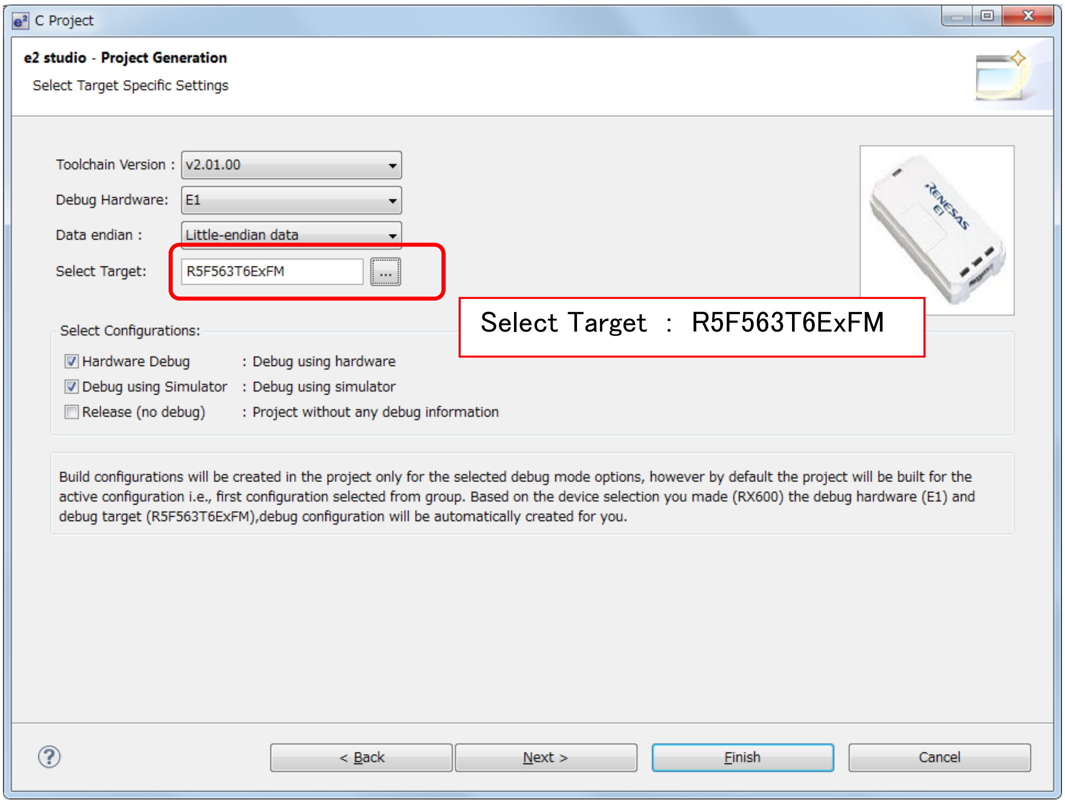


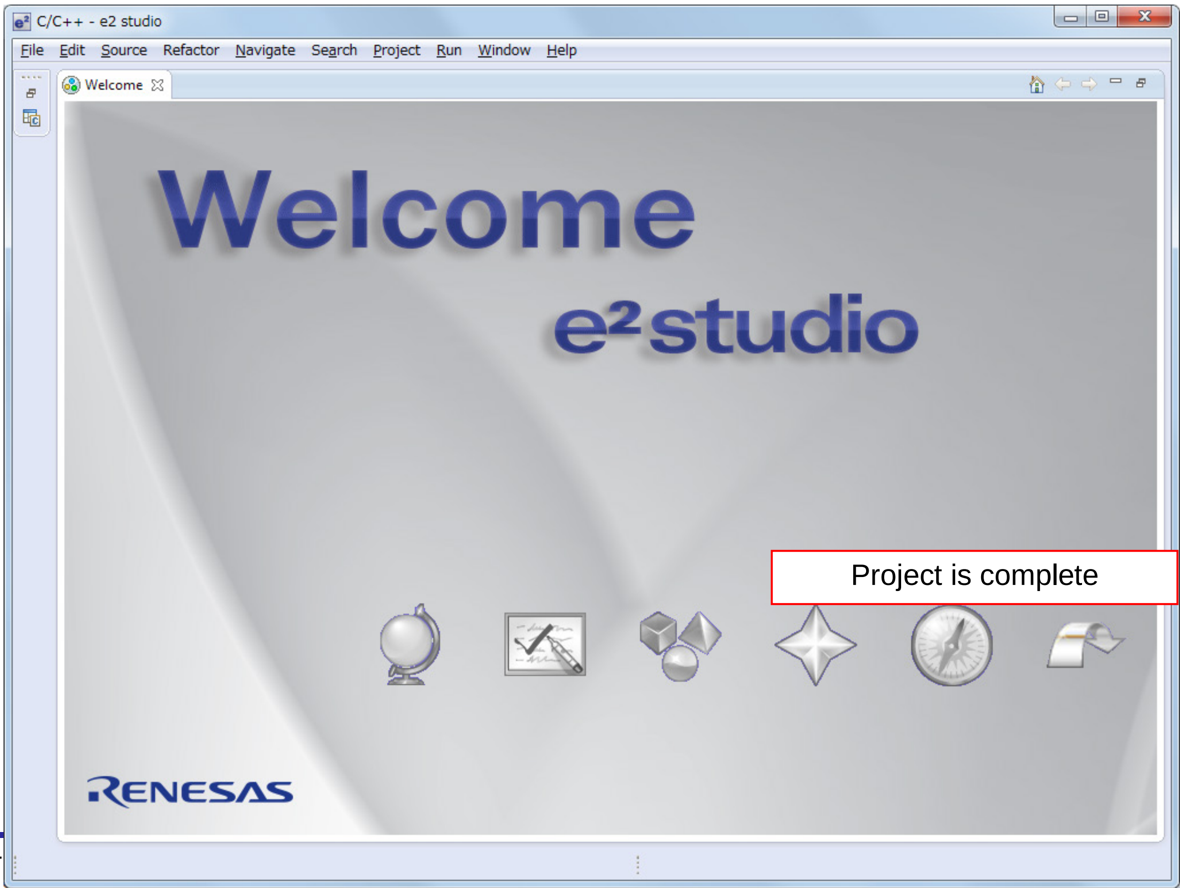
Note : If there are switches that enables/disables TXD0 and RXD1 on the RSK board, enable it.

(1) Preparing the e2 studio project **e2 studio**

Start the e2 studio and make RX63T workspace.




C Project
e2 studio - Project Generation
Select Target Specific Settings
Toolchain Version : v2.01.00
Debug Hardware: E1
Data endian : Little-endian data
Select Target: R5F563T6ExFM
Select Target : R5F563T6ExFM
Select Configurations:
Hardware Debug : Debug using hardware
Debug using Simulator : Debug using simulator
Release (no debug) : Project without any debug information
Build configurations will be created in the project only for the selected debug mode options, however by default the project will be built for the active configuration i.e., first configuration selected from group. Based on the device selection you made (RX600) the debug hardware (E1) and debug target (R5F563T6ExFM),debug configuration will be automatically created for you.
< Back
Next >
Finish
Cancel


C/C++ - e2 studio
File Edit Source Refactor Navigate Search Project Run Window Help
Welcome
Welcome
e²studio
Project is complete
RENESAS

(2) Making the PDG project

PDG

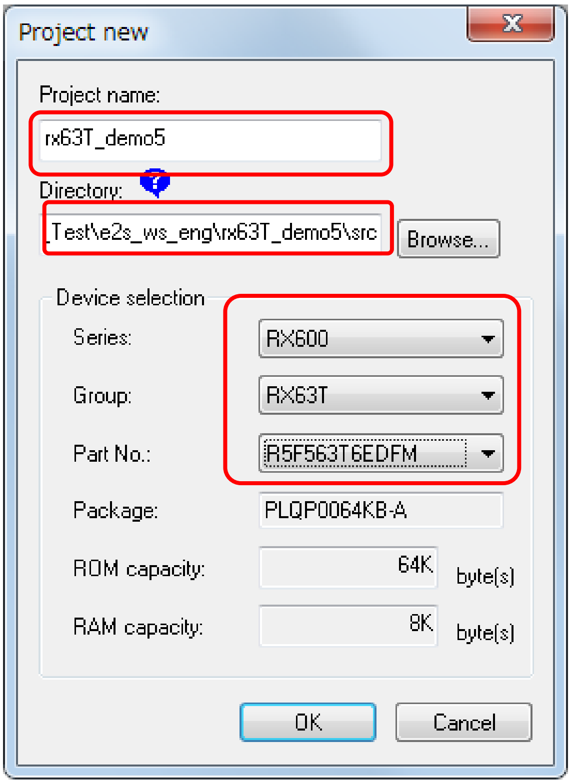
Make the new PDG project “rx630_demo5”. For details on how to make the new PDG project, refer to section 4.1.1 (1), Making the PDG project.

Set the CPU type as follows.

Series : RX600
Group : RX63T
Part No. : R5F563T6EDFM

Note1: If another type of chip is mounted on your RSK board, select corresponding CPU type.

Note2: When making them cooperate with e2 studio, please choose the hierarchy below the src folder of a project of e2 studio by designation of a directory.



(3) Clock setting **PDG**

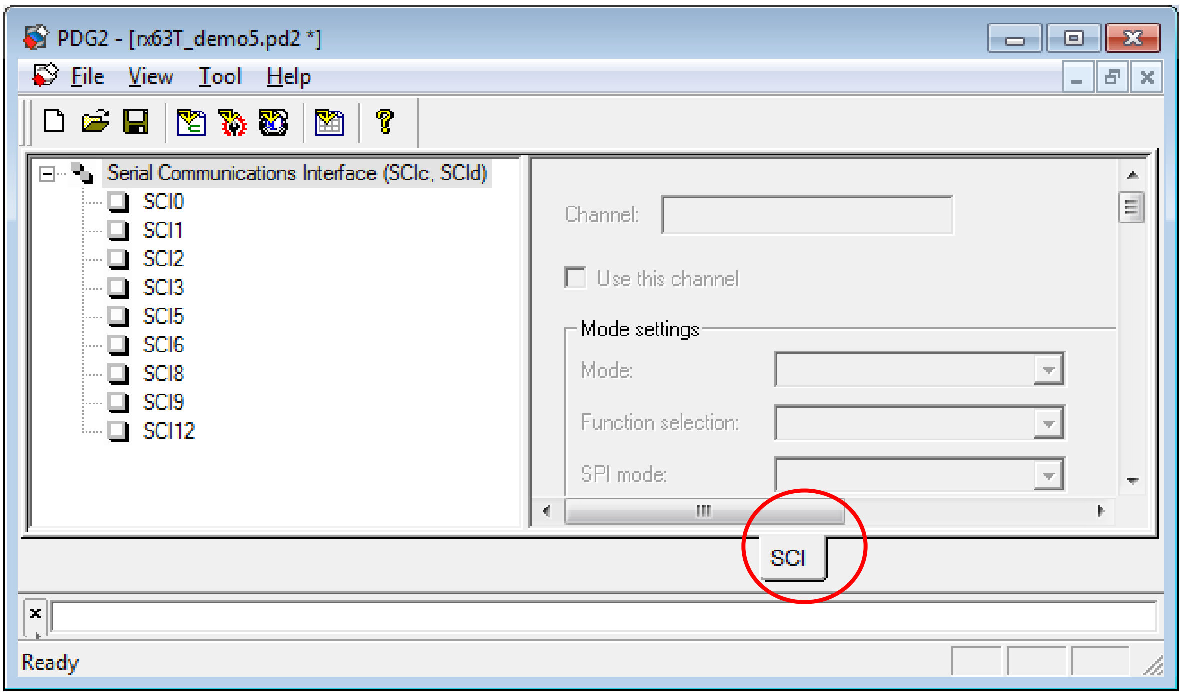
1. The clock setting window opens and the error icons are displayed in the initial state. For icons such as and displayed on window, refer to section 4.1.1 (2), Initial state.
2. For the clock setting, refer to section 4.1.1 (3), Clock setting.

(4) Endian setting **PDG**

For the endian setting, refer to section 3.3, Endian.

(5) SCIc setting **PDG**

Select “SCI” tab to open the SCIc setting window.

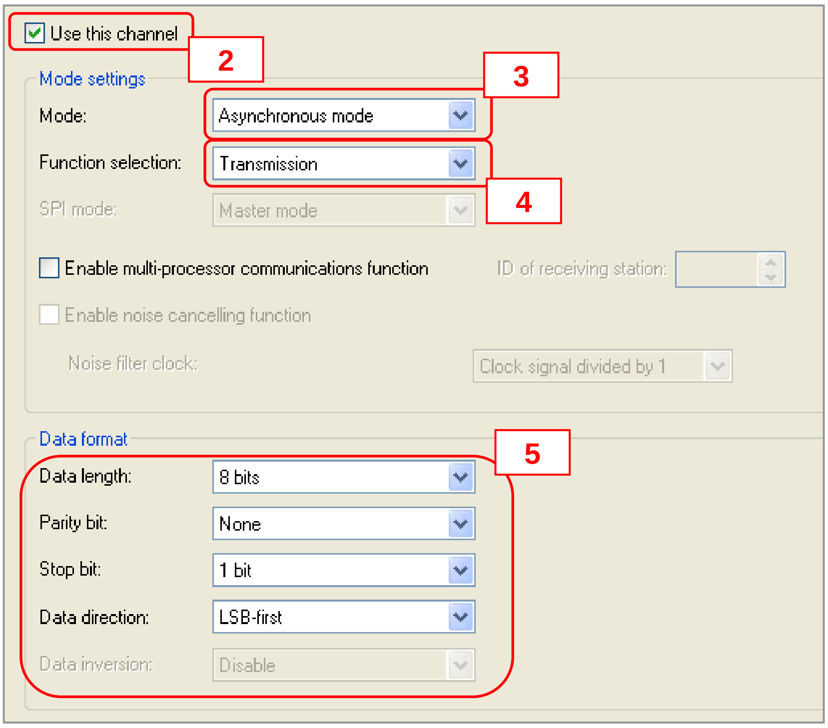


(6) SCI0 (transmitter) setting **PDG**

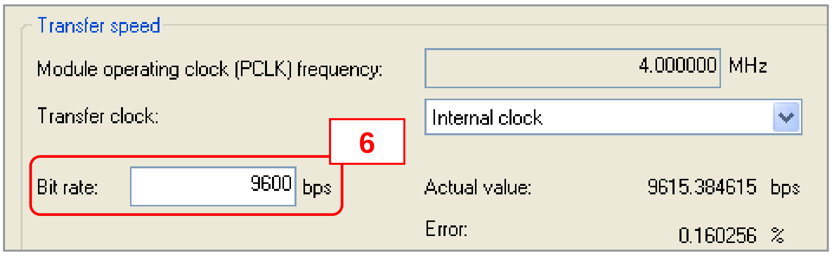
Make the setting for SCI0 as follows.

1. Select SCI0 on the tree view.

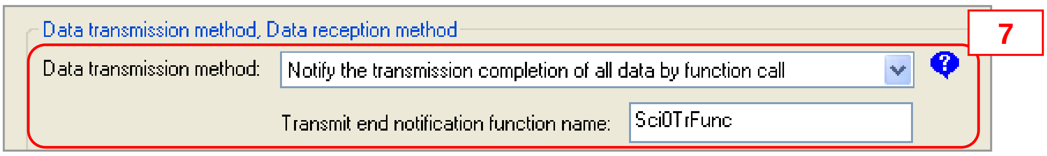
2. Check "Use this channel".
3. Select "Asynchronous mode".
4. Select "Transmission" for the function.
5. Leave the data format settings at the default.



6. Set the bit rate to "9600bps".



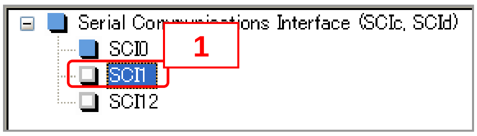
7. Select "Notify the transmission completion of all data by function call" for the data transmission method.



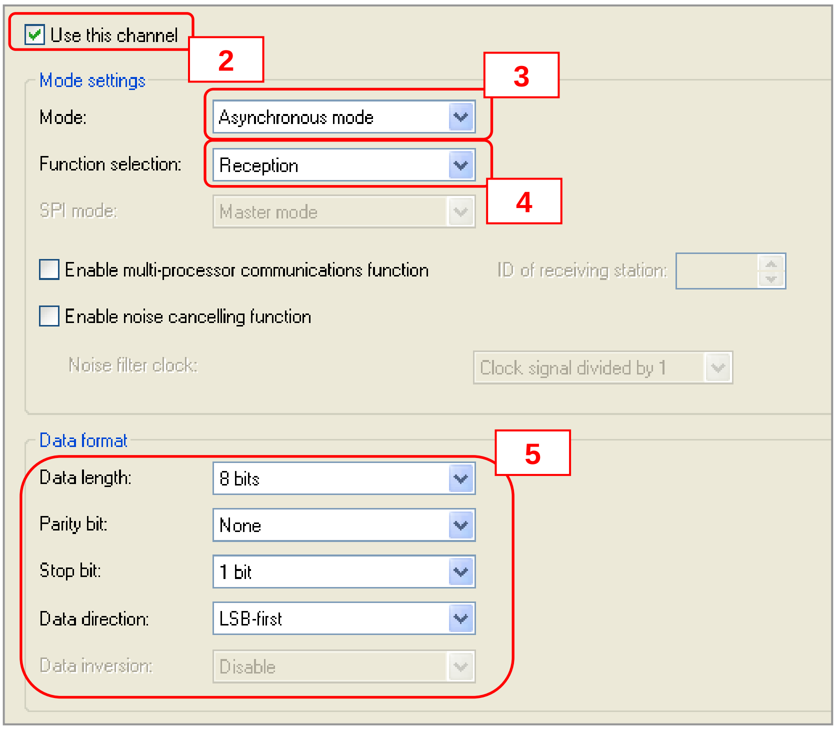
(7) SCI1 (receptor) setting **PDG**

Make the setting for SCI1 as follows.

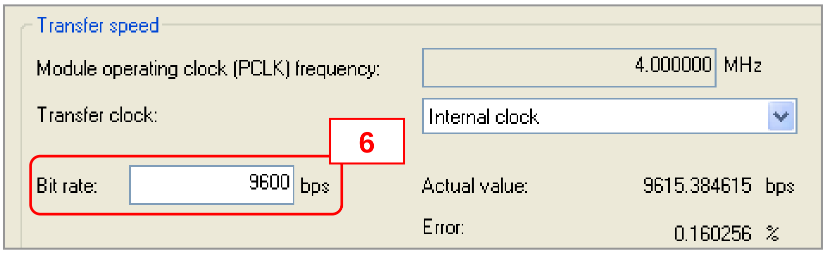
1. Select SCI1 on the tree view.



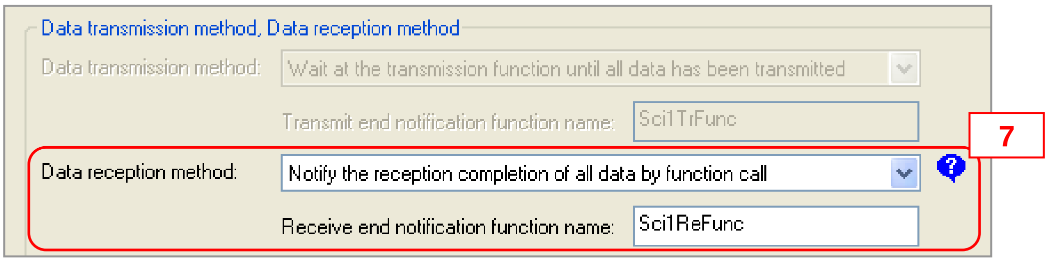
2. Check “Use this channel”.
3. Select “Asynchronous mode”.
4. Select “Reception” for the function.
5. Leave the data format settings at the default.



6. Set the bit rate to “9600bps”.



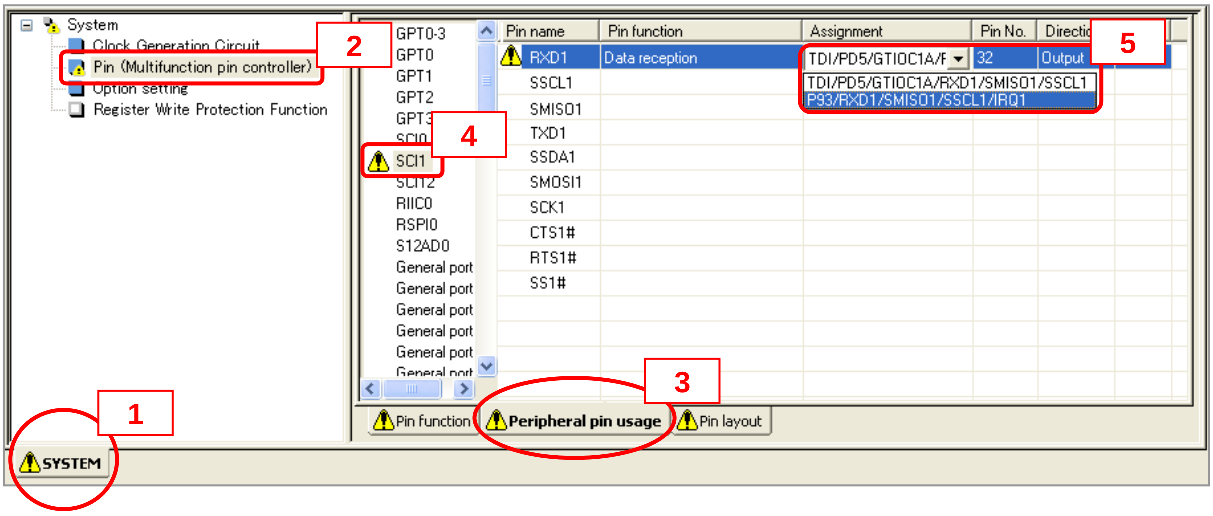
7. Select “Notify the reception completion of all data by function call” for the data reception method.



(8) Pin setting PDG

RXD1 can be selected the pin function assignment. Select the pin function assignment as follows.

1. Select “SYSTEM” tab.
2. Select “Pin(Multifunction pin controller)” on tree view.
3. Select “Perifheral pin usage” tab.
4. Select “SCI1” from the perifheral module list.
5. When the mouse pointer is placed on “assignment” column of RXD1 line, a dropdown button is displayed. Select “P93/RXD1/SMISO1/SSCL1/IRQ1” from the dropdown list.



(9) Generating source files PDG

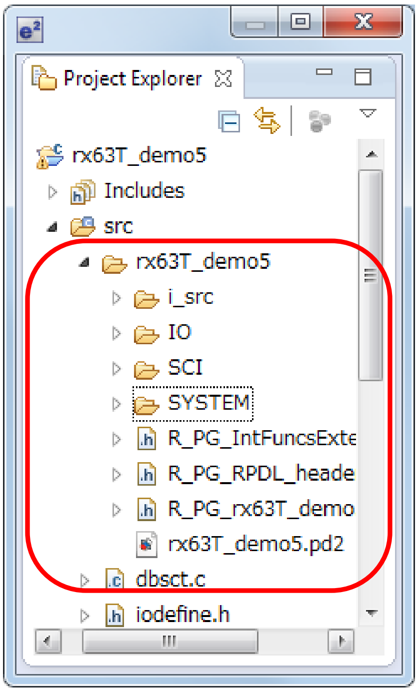
To generate source files, click on the tool bar. For details on generating source files, refer to section 4.1.1 (9), Generating source files.

(10) Adding the generated source files to the e2 studio project **PDG**

To set a build property of e2 studio, click  on the tool bar. A project is established besides the registration of a file. Please refer to "6 About registration to IDE of a generation file" about setting of a project.

A file is added to the project of e2 studio.
An added file is registered by a folder image of a generation source of PDG.

e2 studio



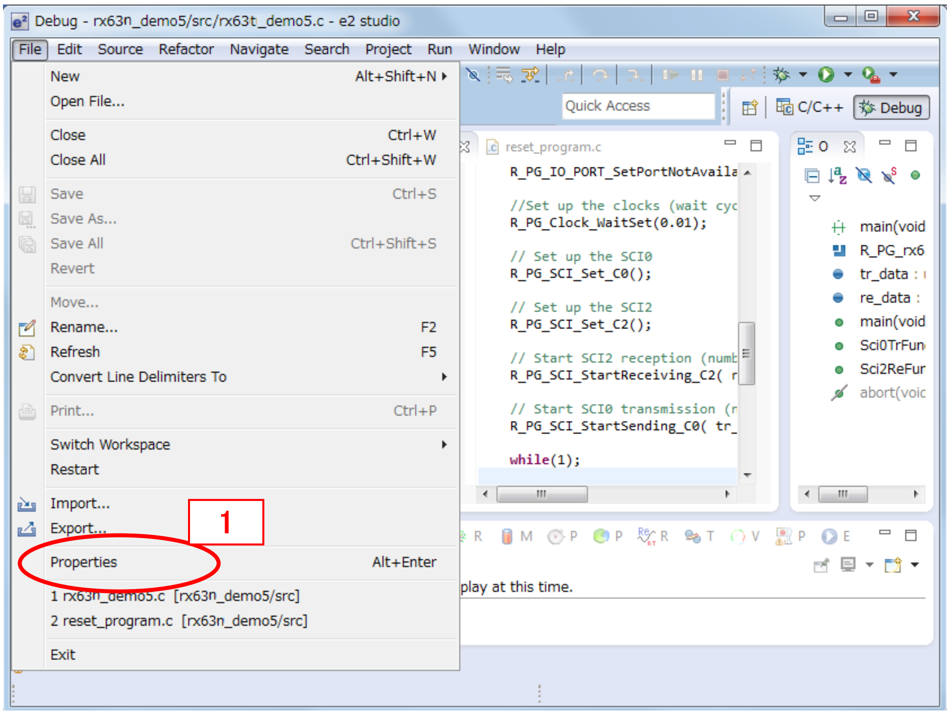
(11) Making the program on HEW **e2 studio**

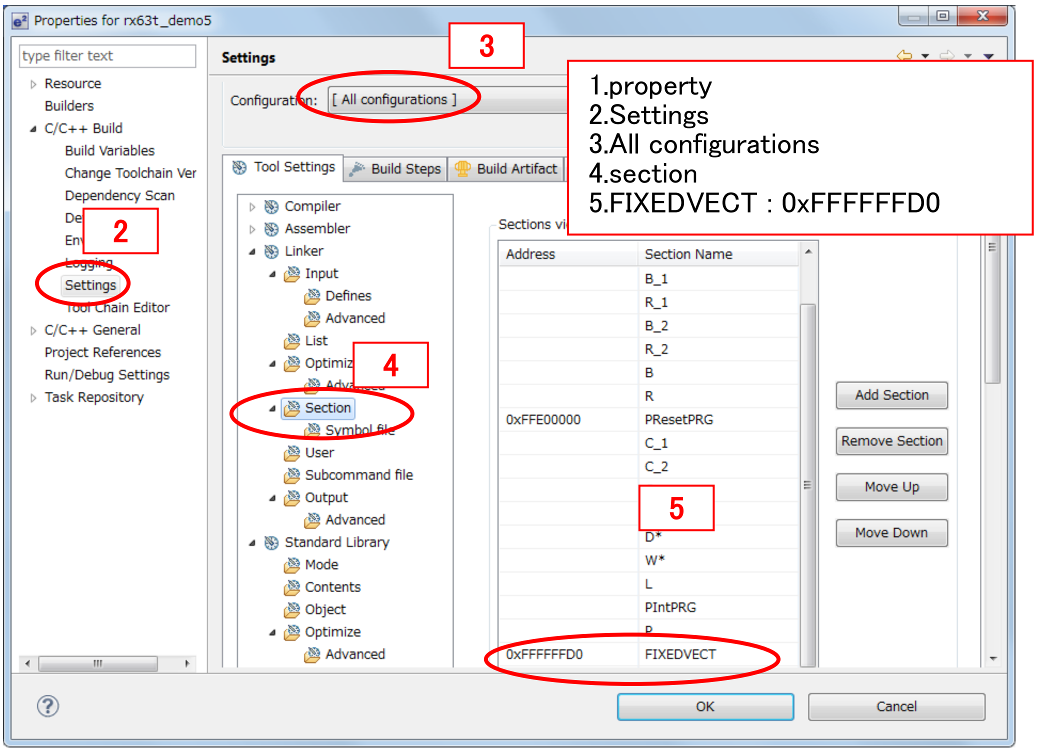
By changing the part of “main” function, make the following program on e2 studio.

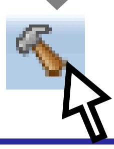
```
//Include "R_PG_<PDG project name>.h"
#include "R_PG_rx63t_demo5.h"

//SCI0 transmission data
uint8_t tr_data[10] = "ABCDEFGHIJ";

//SCI2 reception data storage area
uint8_t re_data[10] = "----------";

void main(void)
{
	//Configure I/O port pins that are not available
	R_PG_IO_PORT_SetPortNotAvailable();

	//Set up the clocks (wait cycle insertion)
	R_PG_Clock_WaitSet(0.01);

	// Set up the SCI0
	R_PG_SCI_Set_C0();

	// Set up the SCI1
	R_PG_SCI_Set_C1();

	// Start SCI1 reception (number of data : 10)
	R_PG_SCI_StartReceiving_C1( re_data, 10 );

	// Start SCI0 transmission (number of data : 10)
	R_PG_SCI_StartSending_C0( tr_data, 10 );

	while(1);
}

//SCI0 transmission end notification function
void Sci0TrFunc(void)
{
	//Stop SCI0 communication
	R_PG_SCI_StopCommunication_C0();
}

//SCI1 reception end notification function
void Sci1ReFunc(void)
{
	//Stop SCI1 communication
	R_PG_SCI_StopCommunication_C1();
}
```

(12) Connecting to the emulator, building the program and downloading **e2 studio**

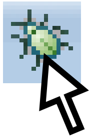
1.Set options options and execute a build.





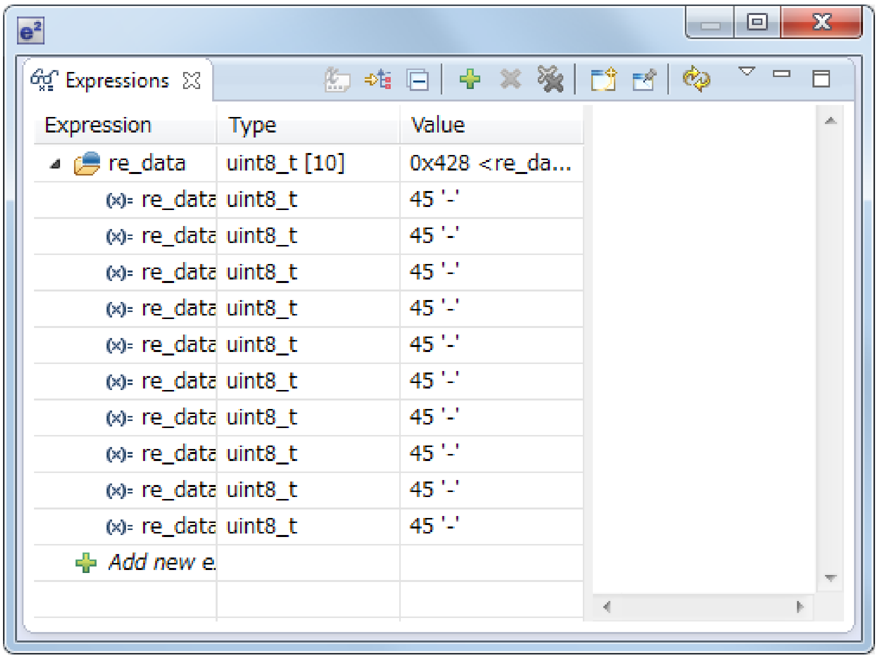


2.download a program.



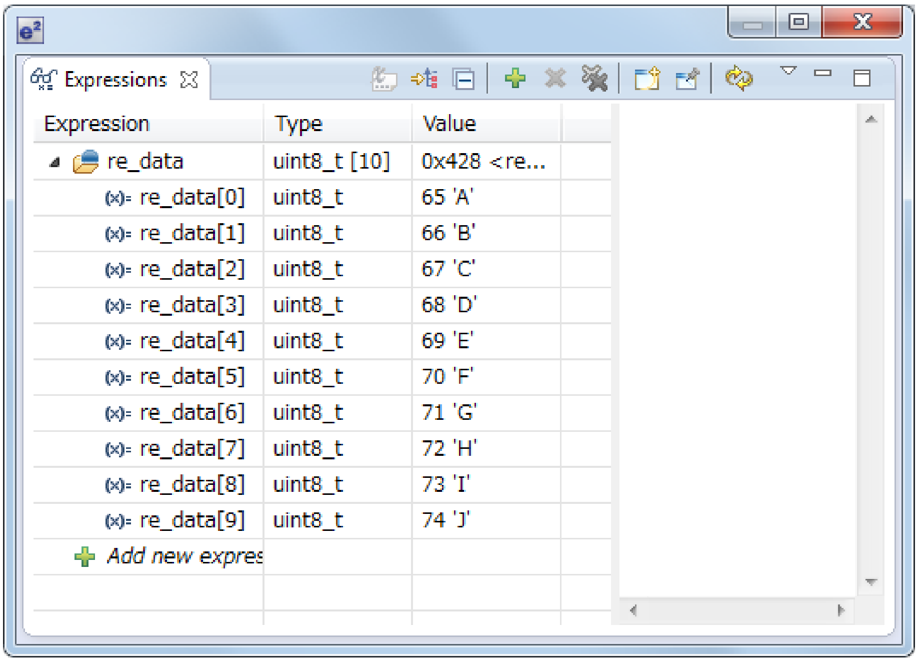
(13) Adding the variable of the reception data **e2 studio**

Open the Expressions window and add the variable "re_data".



(14) Executing the program and monitoring the result of the transfer **e2 studio**

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



4.4 When the HEW is in Use with Renesas starter kit for RX63T(144-pin)

This section introduces the usage of the PDG by giving instructions on how to use the PDG and HEW to create a tutorial program that implements the following operations on the Renesas Starter Kit board for the RX63T(64-pin).

- An LED blinking on a PWM output of the multi-function timer pulse unit 3 (MTU3)
- Continuously scanning on 12-Bit A/D converter (S12ADB)
- Triggering DTCa by ICUb
- Data transfer between SCIc channels 0 and 1

The labels given below respectively indicate operations to take place in the PDG and in the HEW.

PDG	: Operations in the PDG
HEW	: Operations in the HEW

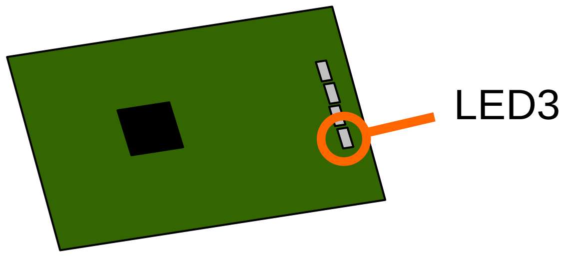
[Note on Using the HEW]
Refer to the user's manual and check the HewTargetServer settings.

4.4.1 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.

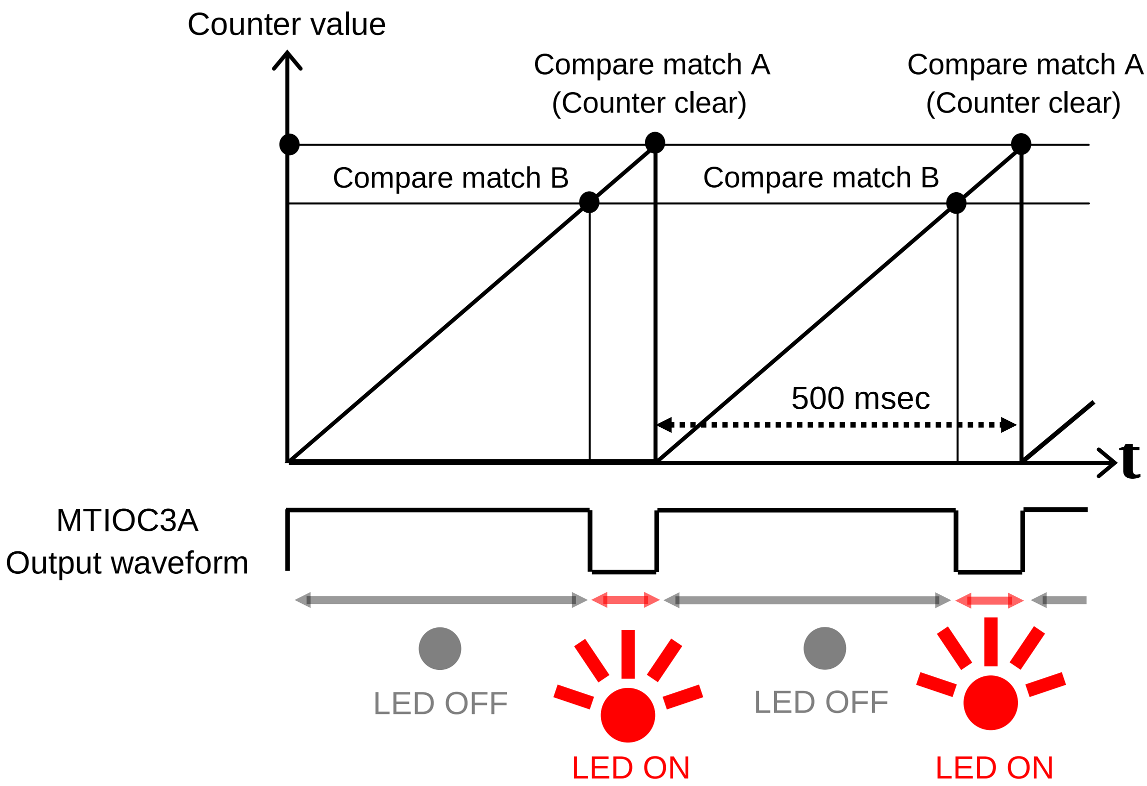
Note : If there is a switch that enables/disables P33(MTIOC3A) on the RSK board, enable it.

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



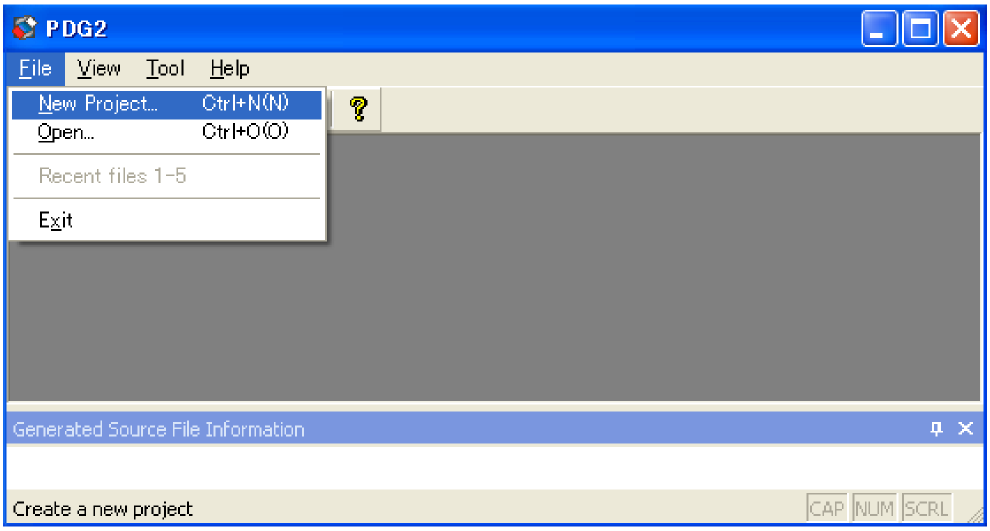
The MTU3 channel 3 (MTU3) will be operated in PWM mode 1. In PWM mode 1, the output signal is controlled by compare match A and B.

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 500 msec)



(1) Making the PDG project **PDG**

1. Start the PDG.
2. Select [File]->[New Project] menu.

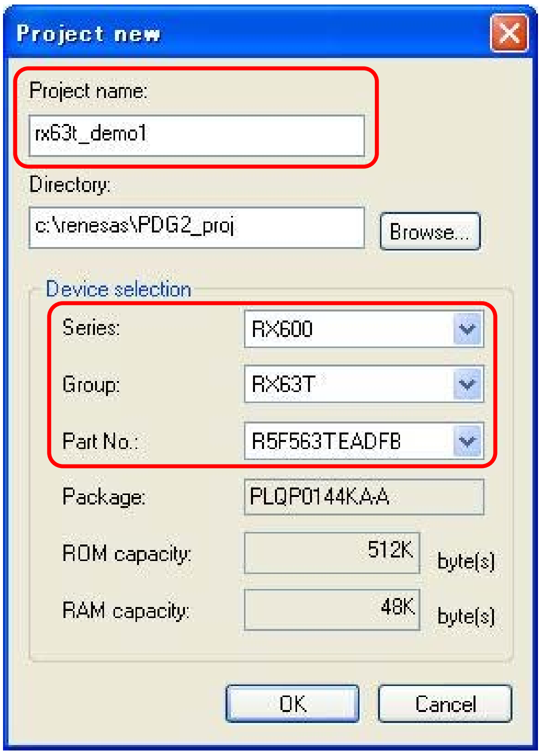


3. Specify "rx63t_demo1" as the project name.

   Set the CPU type as follows.

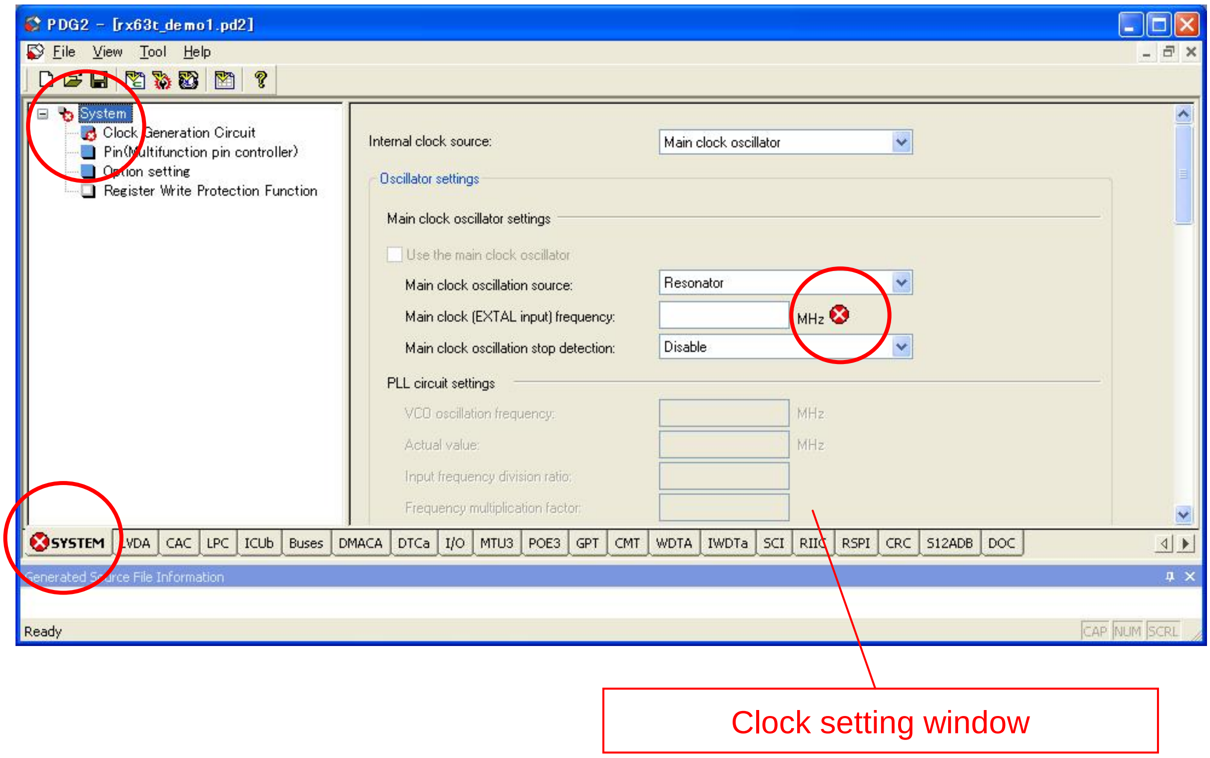
   Series : RX600
   Group : RX63T
   Part No. : R5F563TEADFB

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

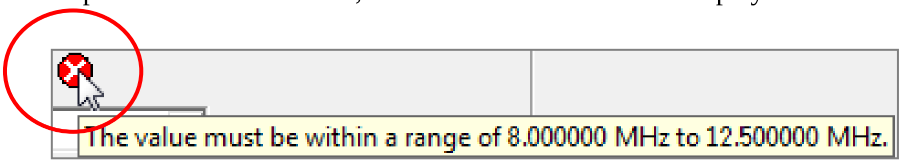


(2) Initial state **PDG**

-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 PDG

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.

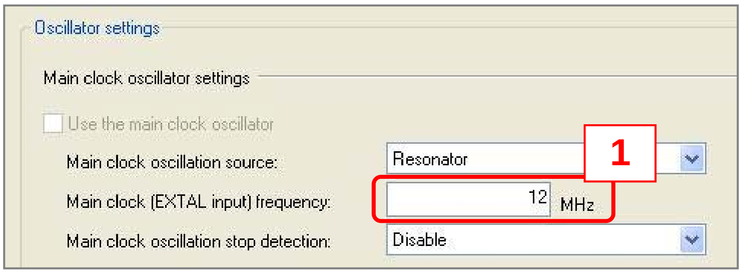
Information

Additional information for the complex setting.

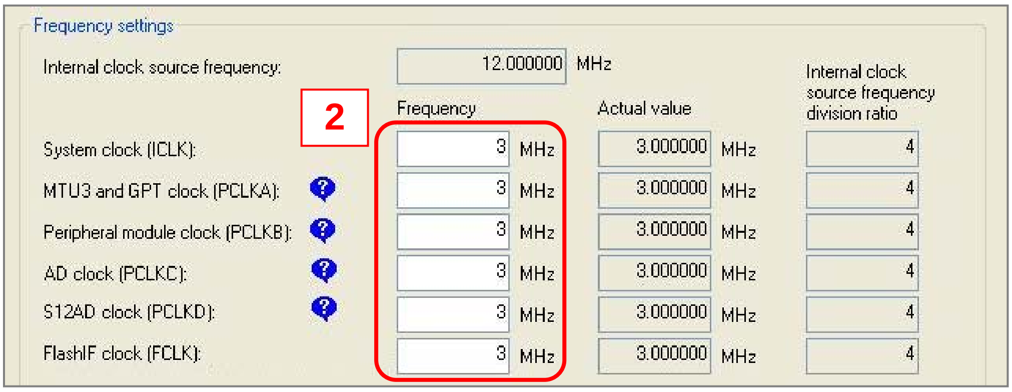
Only icons on the setting window can display the tooltip.

(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 MHz. Set 12 to the edit box.



2. ICLK, PCLKA, PCLKB, PCLKC, PCLKD and FCLK are used in 3 MHz.
   Set 3 to the edit box.



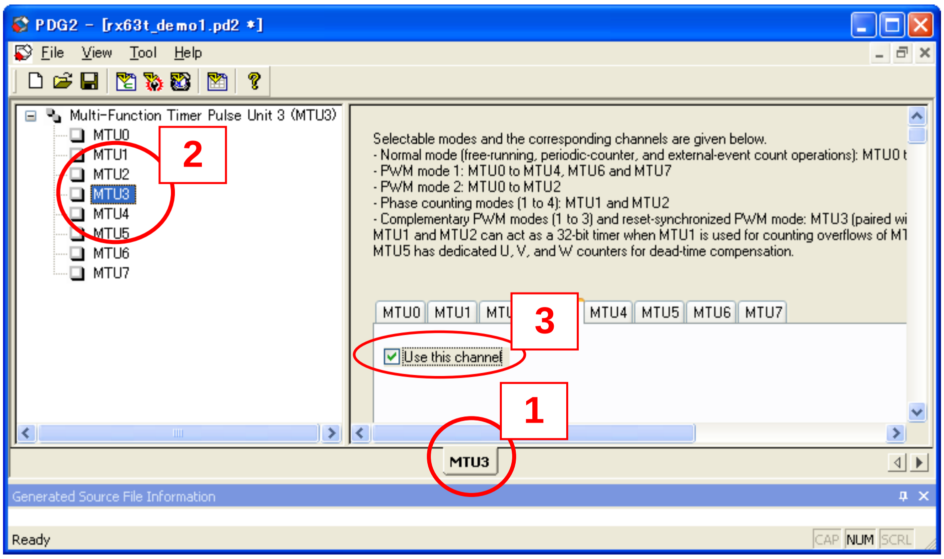
(4) Endian setting **PDG**

For the endian setting, refer to section 3.3, Endian.

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

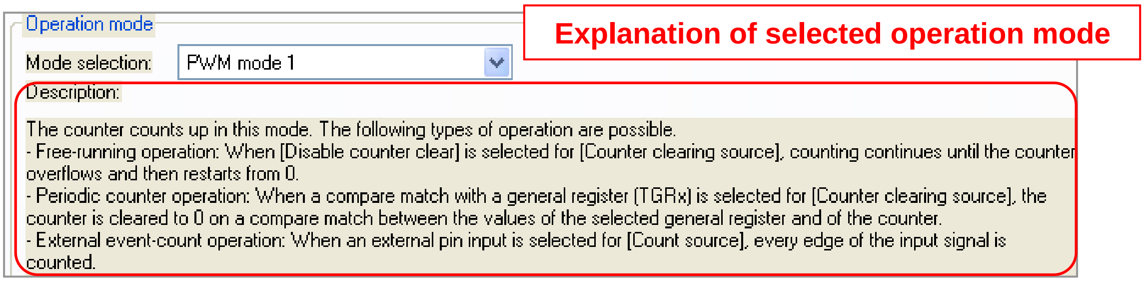
Opening MTU3 channel 3(MTU3) setting window.

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



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

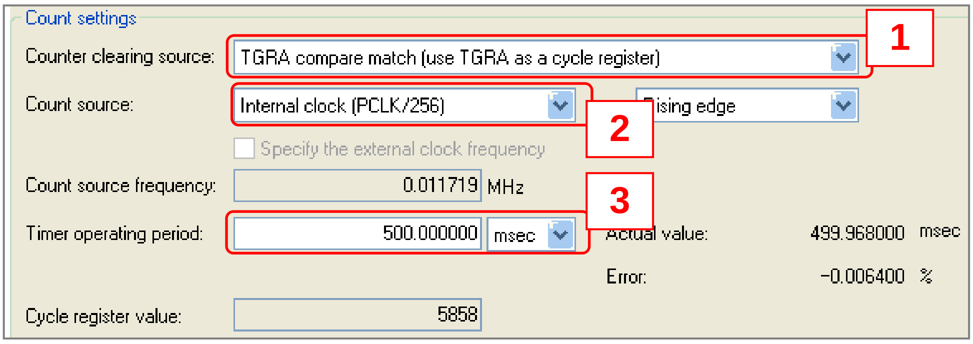
Select “PWM mode 1” for the operation mode.



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

The counter setting is as follows.

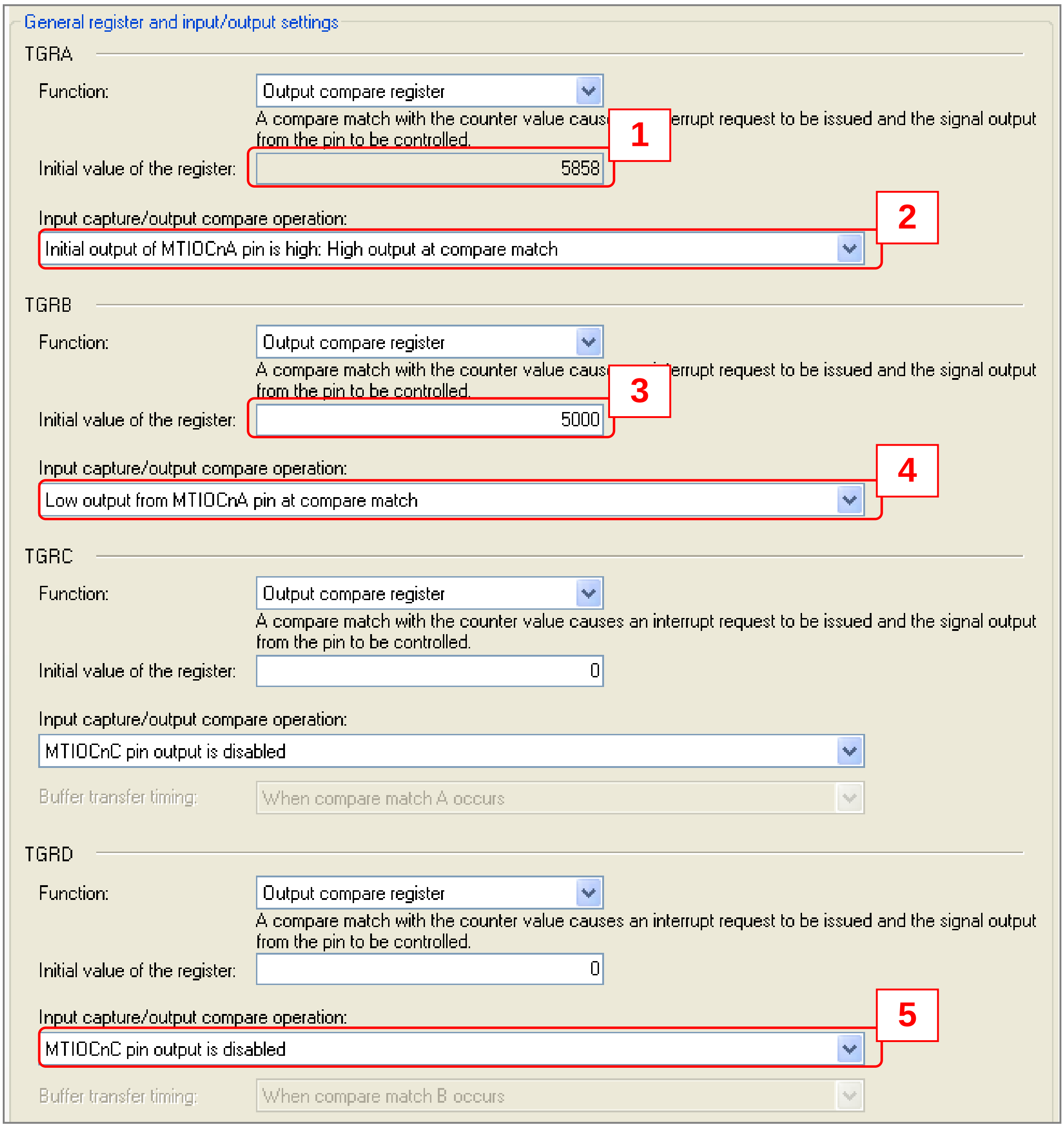
1. Select “TGRA compare match” for a counter clearing source.
2. Select “Internal clock (PCLK/256)” for a count source.
3. Set “500 msec” to timer operation period.



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

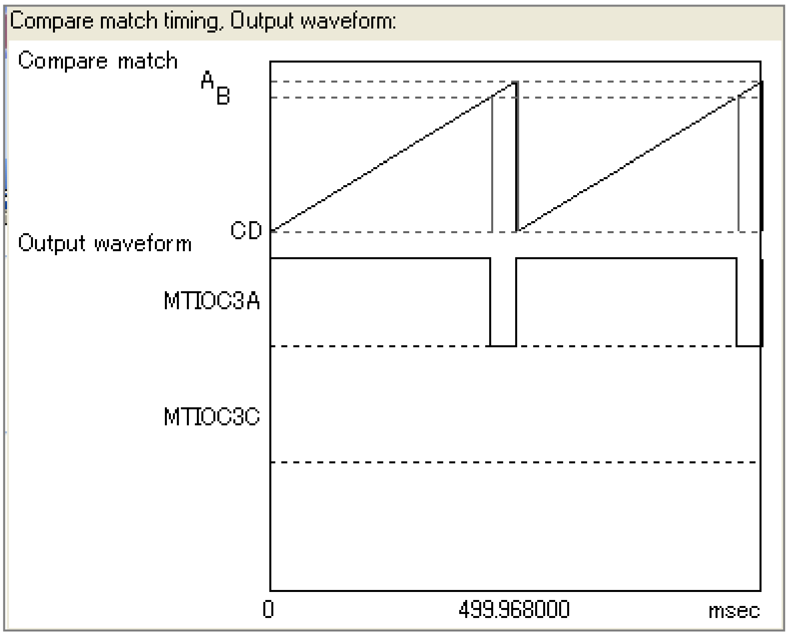
General register setting is as follows.

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 “5000” to TGRB initial value.
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.



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

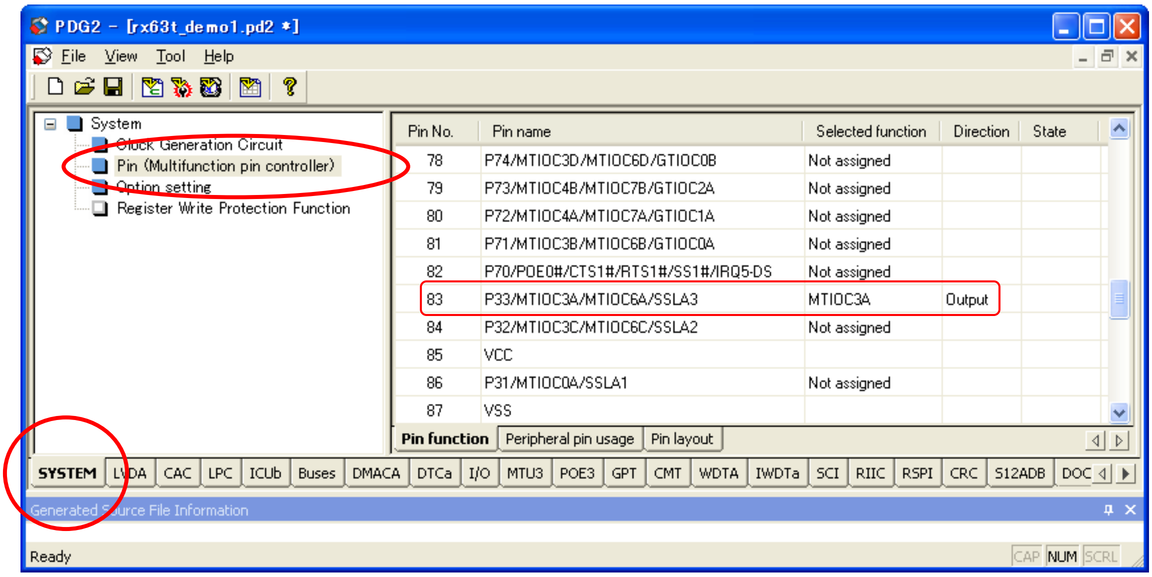
The compare match timing and the output waveform are displayed in a diagram.



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

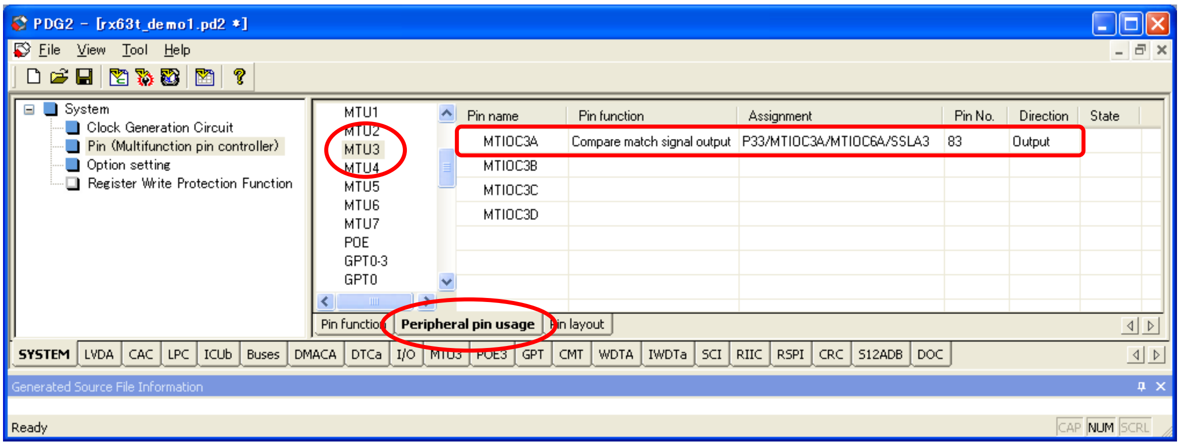
- It is possible to check the usage of pins on the pin function windows

1. After setting up the MTU3, select "SYSTEM" tab and click "Pin (Multifunction pin controller)" on the tree view.
2. On the Pin function window, you can see that No.83 pin is used as MTIOC3A.



- State of pin usage for each peripheral module is displayed in the peripheral pin usage window.

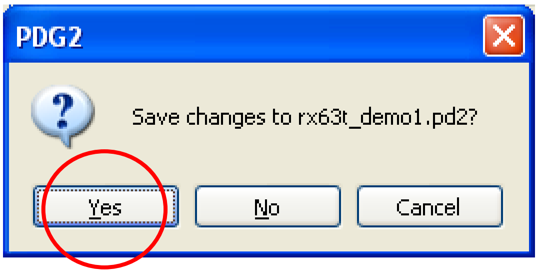
Select peripheral pin usage sheet and click MTU3 to check the usage of MTIOC3A pin.



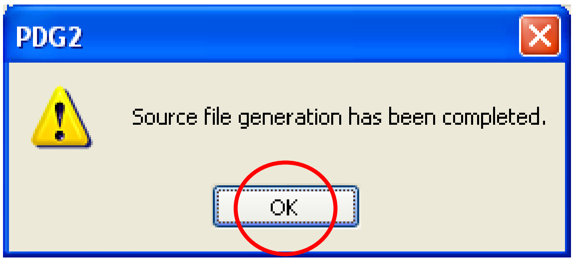
(11) Generating source files **PDG**

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

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

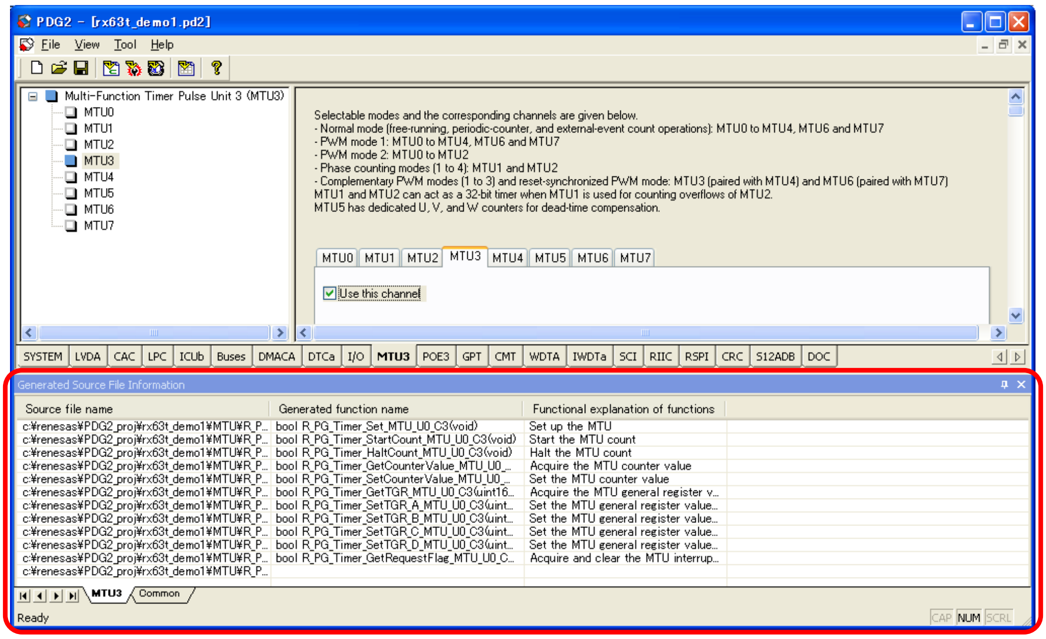


3. Click [OK] on the message box.



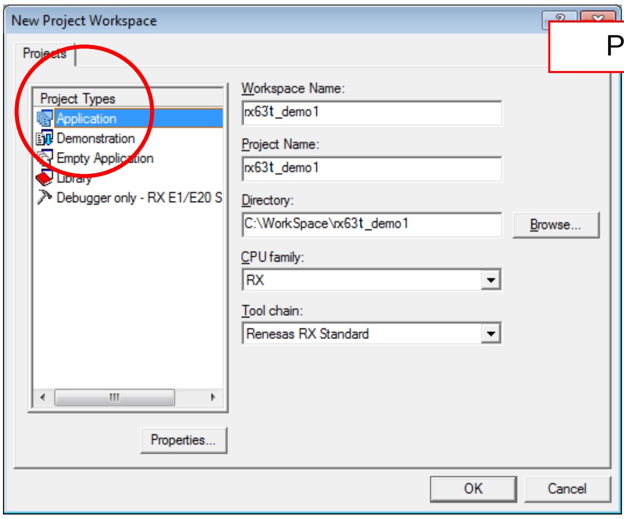
4. Generated functions are listed in lower pane.

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

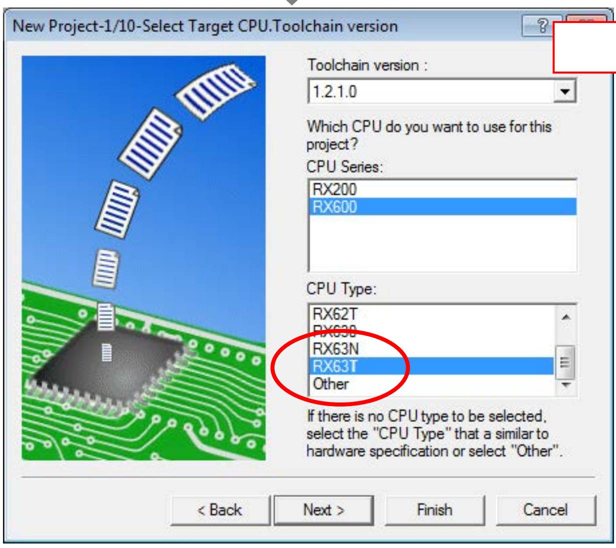


(12) Preparing the HEW project **HEW**

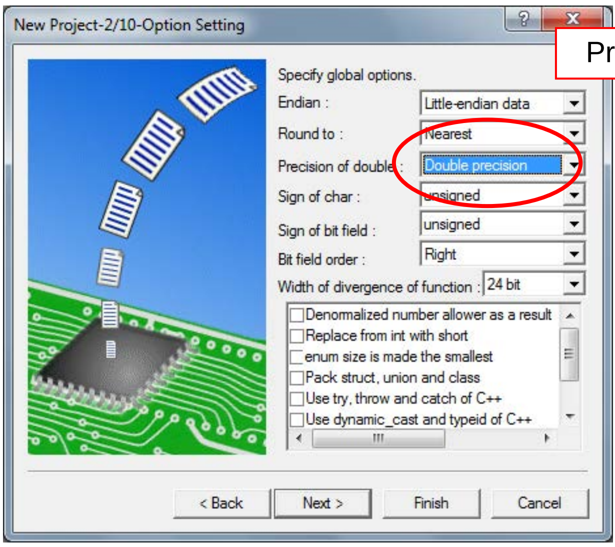
Start the HEW and make RX63T workspace.



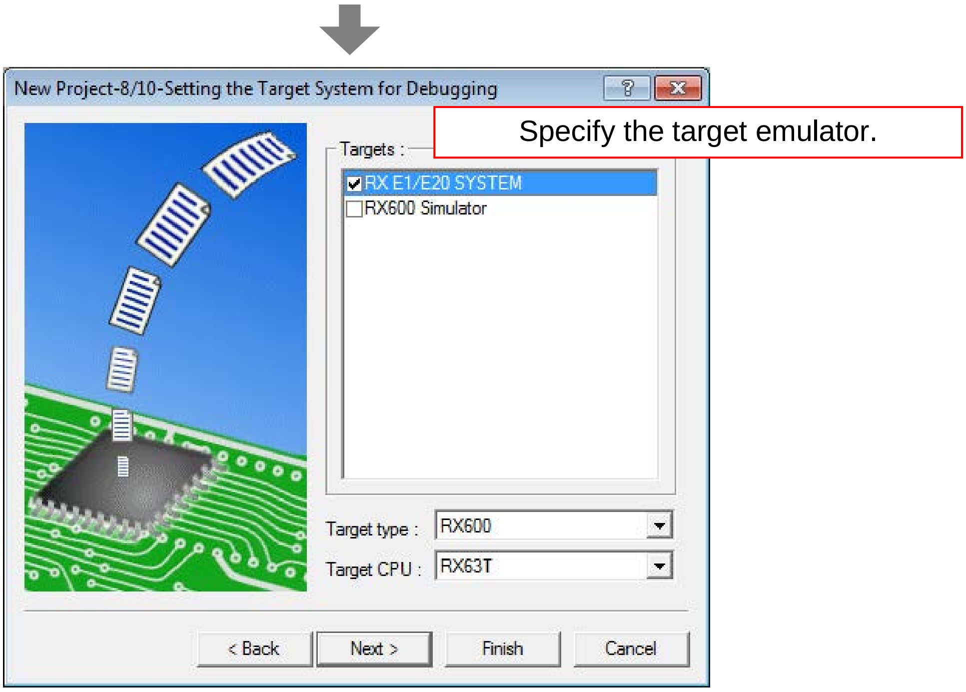
Project type : Application

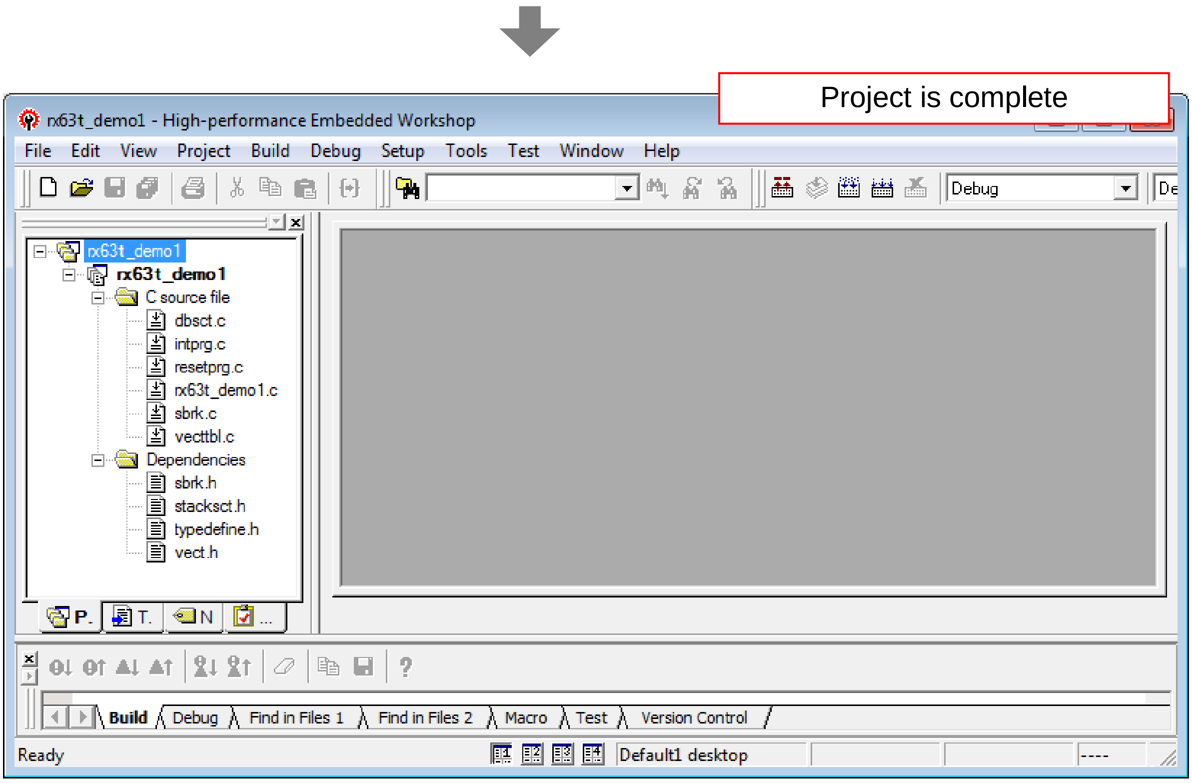


CPU type : RX63T



Precision of double : Double precision


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 : RX63T
< Back
Next >
Finish
Cancel

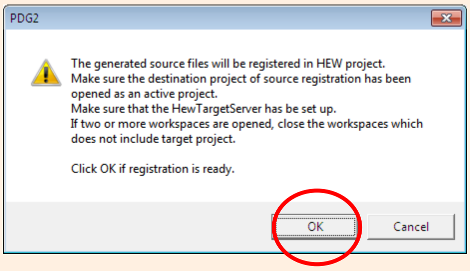

rx63t_demo1 - High-performance Embedded Workshop
Project is complete
File Edit View Project Build Debug Setup Tools Test Window Help
Debug
rx63t_demo1
rx63t_demo1
C source file
dbsct.c
intprg.c
resetprg.c
rx63t_demo1.c
sbrk.c
vecttbl.c
Dependencies
sbrk.h
stacksct.h
typedefine.h
vect.h
Build Debug Find in Files 1 Find in Files 2 Macro Test Version Control
Ready
Default1 desktop

(13) Adding the generated source files to the HEW project

PDG

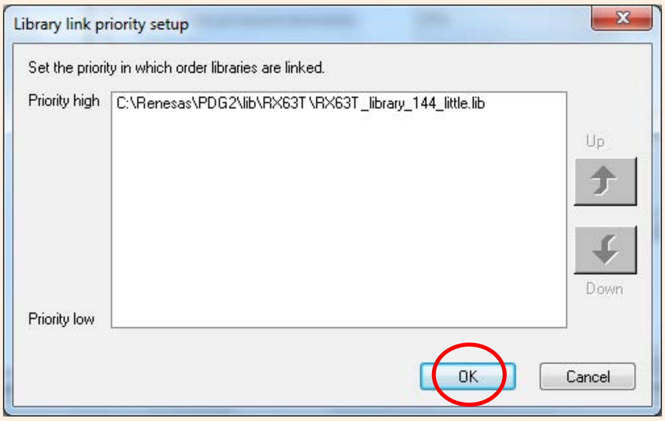
1. To add source files to HEW, click  on the tool bar.

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



3. This is a linkage setting of RPDL library.

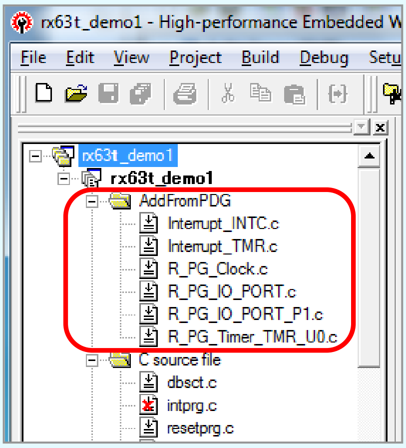
When using multiple lib files, linkage order can be set in this dialog box.



HEW

4. Source files are added to HEW

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 PDG user's manual.

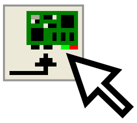
(14) Making the program on HEW **HEW**

By changing the part of “main” function, make the following program on HEW.

```
//Include "R_PG_<PDG project name>.h"
#include "R_PG_rx63t_demo1.h"

void main(void)
{
	//Configure I/O port pins that are not available
	R_PG_IO_PORT_SetPortNotAvailable();

	//Set up the clocks (wait cycle insertion)
	R_PG_Clock_WaitSet(0.01);

	//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);
}
```

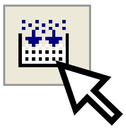
(15) Connecting to the emulator, building the program and executing **HEW**

1. Connect to the emulator



Connect button

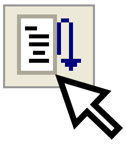
2. Just by clicking [Build] button, program can be built because RPDL library and include directory are automatically registered in build setting.



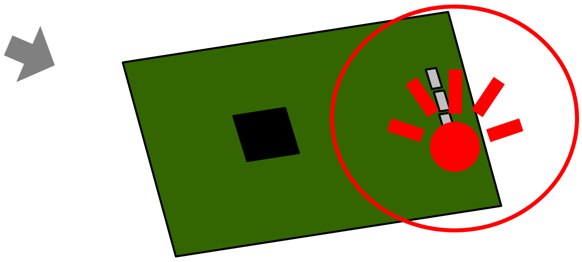
Build button

3. Download the program.

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

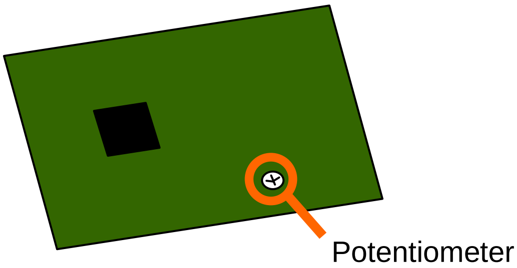


Reset go button



4.4.2 Continuously scanning on 12-Bit A/D converter (S12ADB)

In RX63T RSK board, the potentiometer is connected to AN000 analog input. In this tutorial, the 12-Bit A/D converter (S12ADB) will be set up to execute A/D conversion continuously. And the result of A/D conversion will be monitored on HEW.



Note : If there is a switch that enables/disables AN000 on the RSK board, enable it.

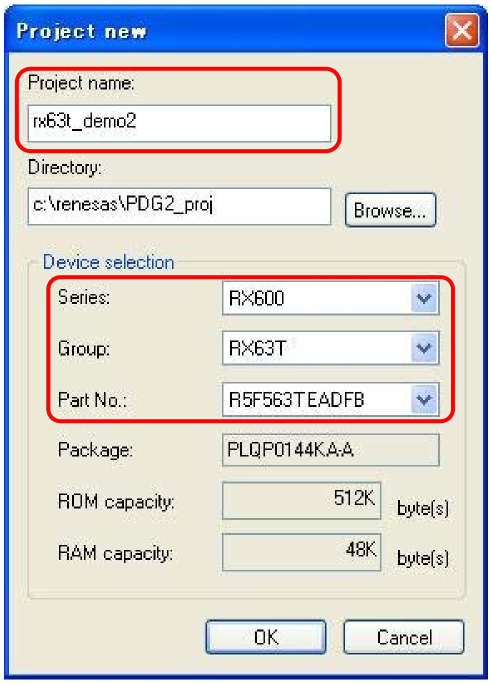
(1) Making the PDG project **PDG**

Make the new PDG project “rx63t_demo2”. For details on how to make the new PDG project, refer to section 4.1.1 (1), Making the PDG project.

Set the CPU type as follows.

Series : RX600
Group : RX63T
Part No. : R5F563TEADFB

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



(2) Clock setting **PDG**

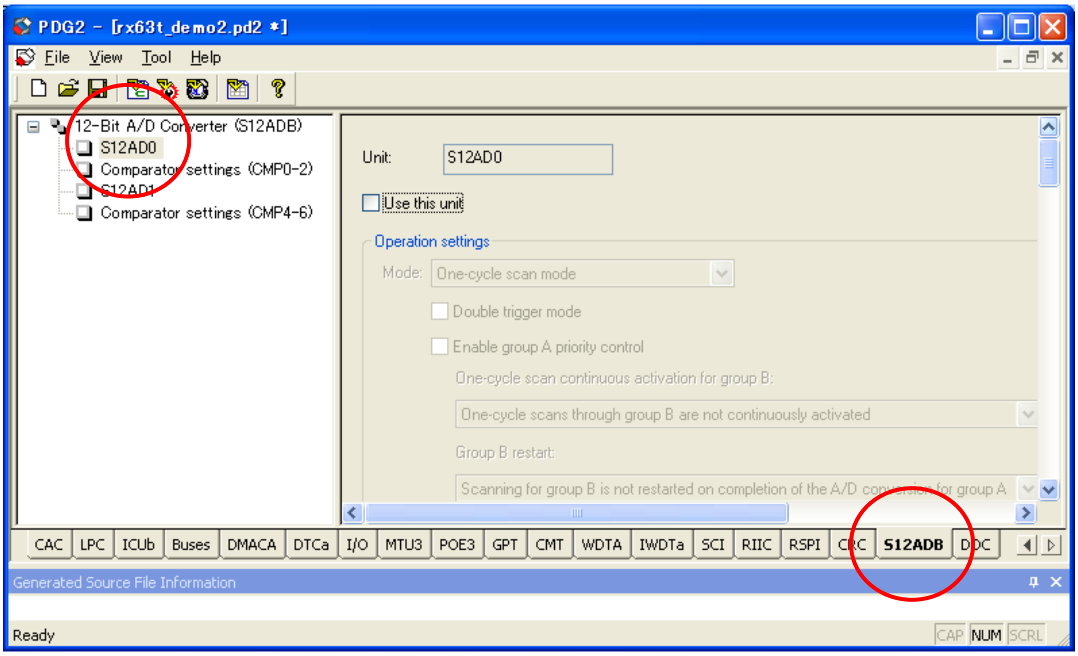
1. The clock setting window opens and the error icons are displayed in the initial state. For icons such as  and  displayed on window, refer to section 4.1.1 (2), Initial state.
2. For the clock setting, refer to section 4.1.1 (3), Clock setting.

(3) Endian setting **PDG**

For the endian setting, refer to section 3.3, Endian.

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

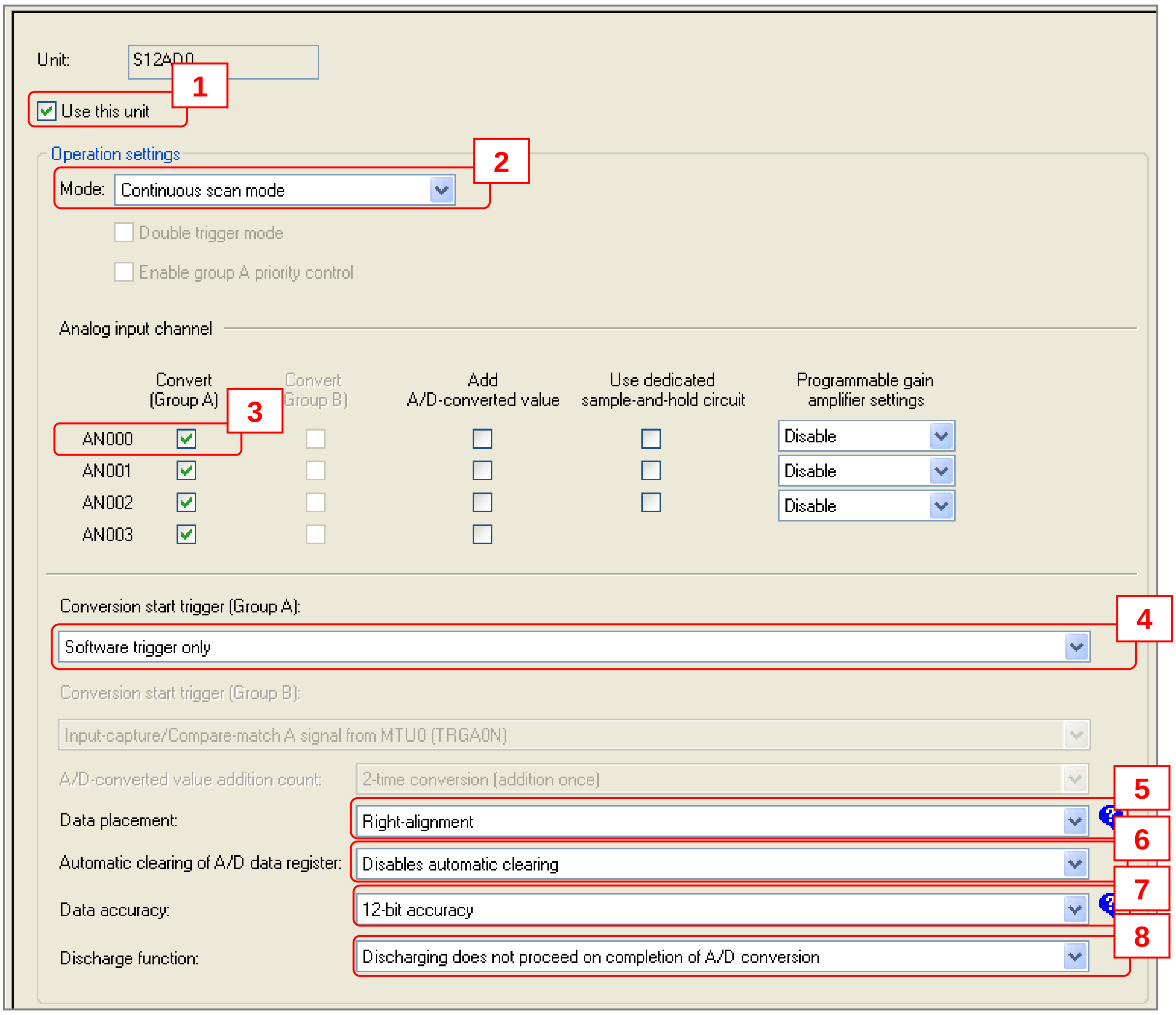
Select "S12ADB" tab and click S12AD0 on tree view.



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

Make the following setting for S12AD0.

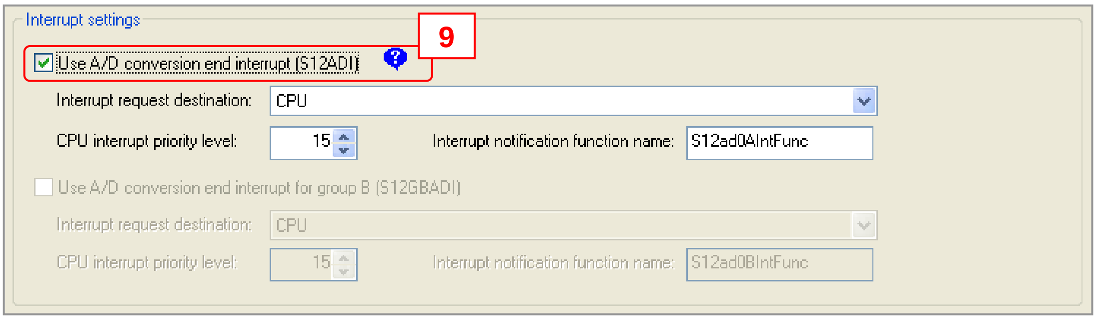
1. Check "Use this unit".
2. Select "Continuous scan mode" for the operation mode.
3. Check "AN000" for the analog input channel.
4. Select "Software trigger only" for the conversion start trigger.
5. Select "Right-alignment" for the data placement.
6. Select "Disables automatic clearing" for the automatic clearing of A/D data register.
7. Select"12-bit accuracy" for the data accuracy.
8. Select "Discharging does not proceed on completion of A/D conversion" for the discharge function.



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

Make the following setting for S12AD0.

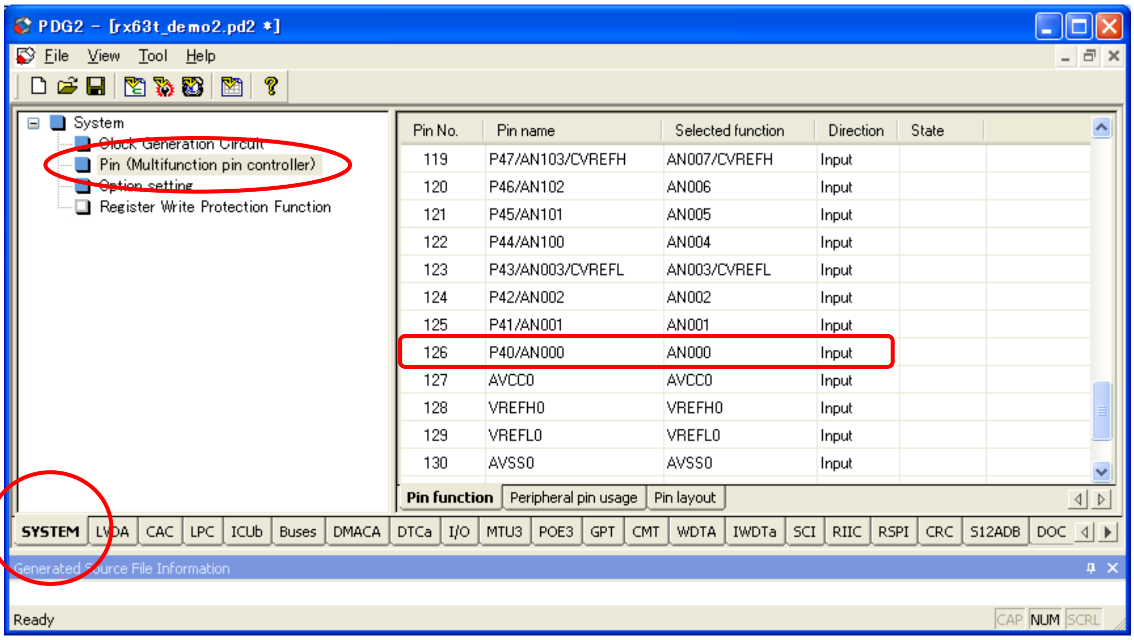
9. Check "Use A/D conversion end interrupt (S12ADI)".



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

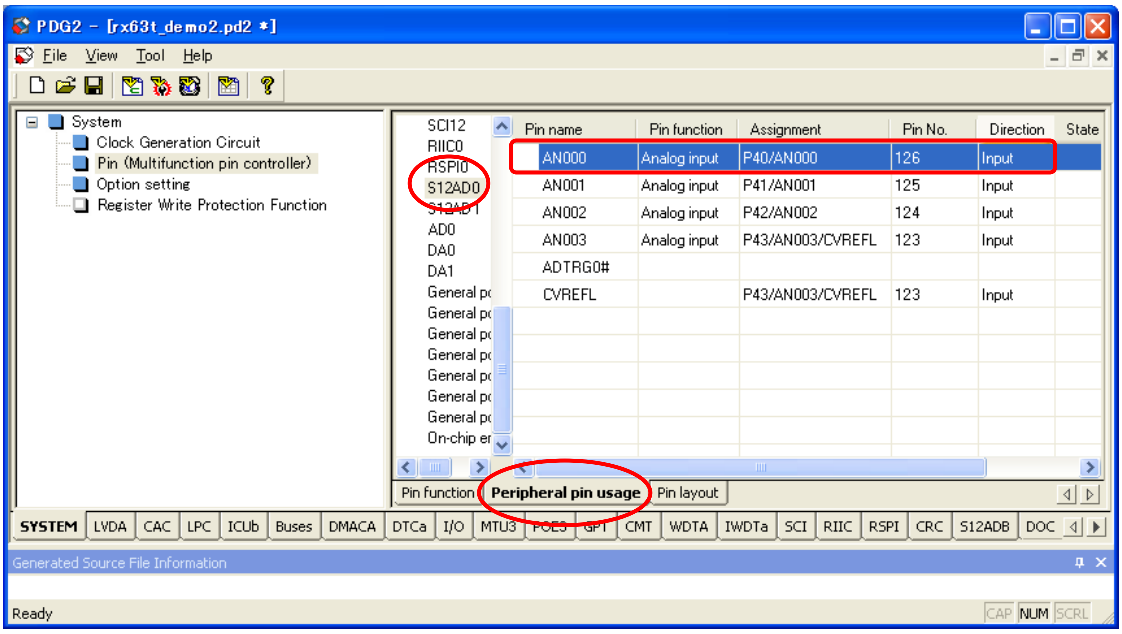
- It is possible to check the usage of pins on the pin function windows

1. After setting up the S12ADB, select “SYSTEM” tab and click “Pin” on the tree view.
2. On the Pin function window, you can see that No.126 pin is used as AN000.



- State of pin usage for each peripheral module is displayed in the peripheral pin usage window.

Select peripheral pin usage sheet and click S12AD0 to check the usage of AN000 pin.



(8) Generating source files **PDG**

To generate source files, click  on the tool bar. For details on generating source files, refer to section 4.1.1 (9), Generating source files.

(9) Preparing the HEW project **HEW**

Start the HEW and make RX63T workspace. For details on making HEW project, refer to section 4.1.1 (10), Preparing the HEW project.

(10) Adding the generated source files to the HEW project **PDG**

To add the generated source files to HEW, click  on the tool bar. For details on adding the source files to HEW project, refer to section 4.1.1 (11), Adding the generated source files to the HEW project.

(11) Making the program on HEW **HEW**

By changing the part of "main" function, make the following program on HEW.

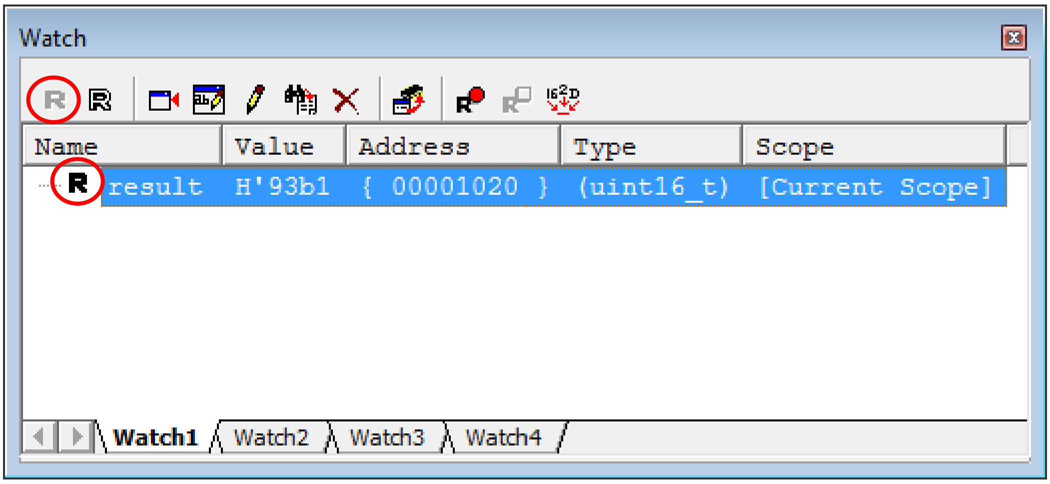
```
//Include "R_PG_<PDG project name>.h"
#include "R_PG_rx63t_demo2.h"

void main(void)
{
    //Set up the clocks (wait cycle insertion)
    R_PG_Clock_WaitSet(0.01);

    //Set up A/D converter
    R_PG_ADC_12_Set_S12AD0();

    //Start A/D conversion
    R_PG_ADC_12_StartConversion_S12AD0();

    while(1);
}

//Variable to store the result
uint16_t result;

//A/D conversion end interrupt notification function
void S12ad0AIntFunc(void)
{
    //Get the result of conversion
    R_PG_ADC_12_GetResult_S12AD0(&result, 0, 0, 0);
}
```

(12) 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 4.1.1 (13), connecting to the emulator, building the program and executing.

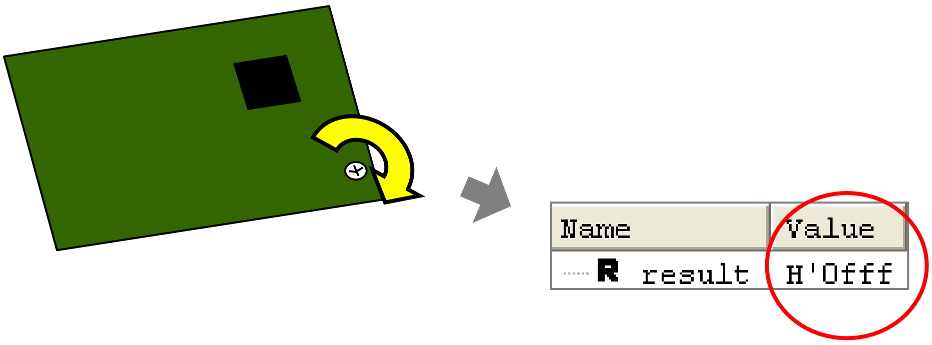
(13) Adding the variable of A/D conversion result to the watch window **HEW**

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



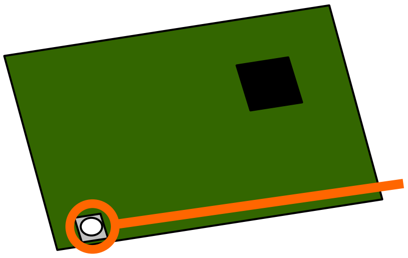
(14) Executing the program and monitoring the A/D conversion result **HEW**

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



4.4.3 Triggering DTCa by ICUb

In RX63T RSK board, switch 1 (SW1) is connected to IRQ0. In this tutorial, the data transfer controller (DTCa) and ICUb will be set up and DTC transfer triggered by IRQ0 will be performed.



Note : If there is a switch that enables/disables IRQ0 on the RSK board, enable it.

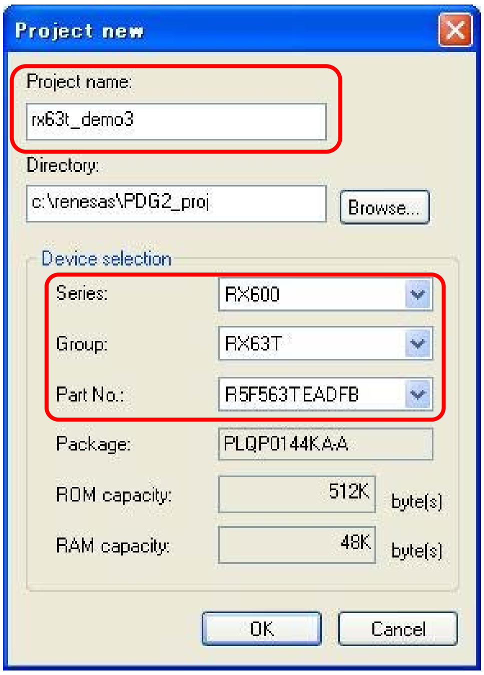
(1) Making the PDG project **PDG**

Make the new PDG project “rx63t_demo3”. For details on how to make the new PDG project, refer to section 4.1.1 (1), Making the PDG project.

Set the CPU type as follows.

Series : RX600
Group : RX63T
Part No. : R5F563TEADFB

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



(2) Clock setting **PDG**

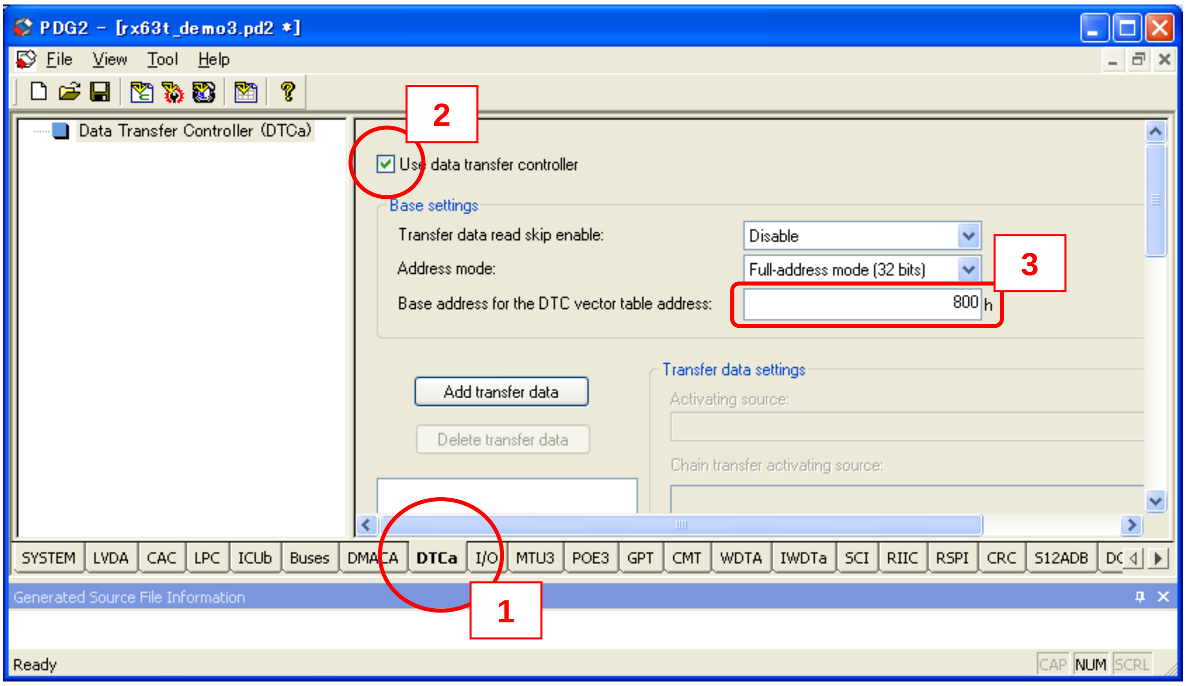
1. The clock setting window opens and the error icons are displayed in the initial state. For icons such as and displayed on window, refer to section 4.1.1 (2), Initial state.
2. For the clock setting, refer to section 4.1.1 (3), Clock setting.

(3) Endian setting **PDG**

For the endian setting, refer to section 3.3, Endian.

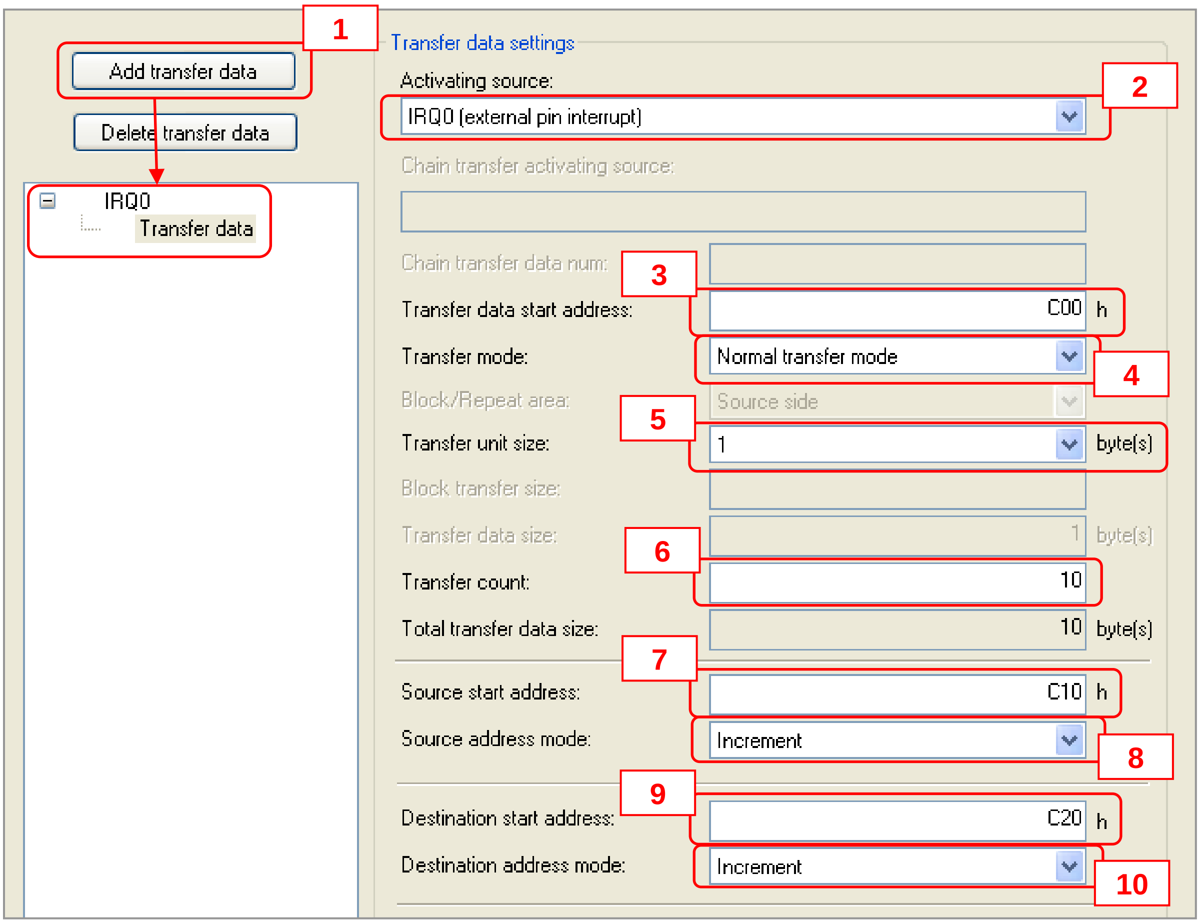
(4) DTCa setting-1 **PDG**

1. Select “DTCa” tab to open the DTCa setting window.
2. Check "Use data transfer controller".
3. The DTCa vector table will be allocated from 800h. Set “800”.



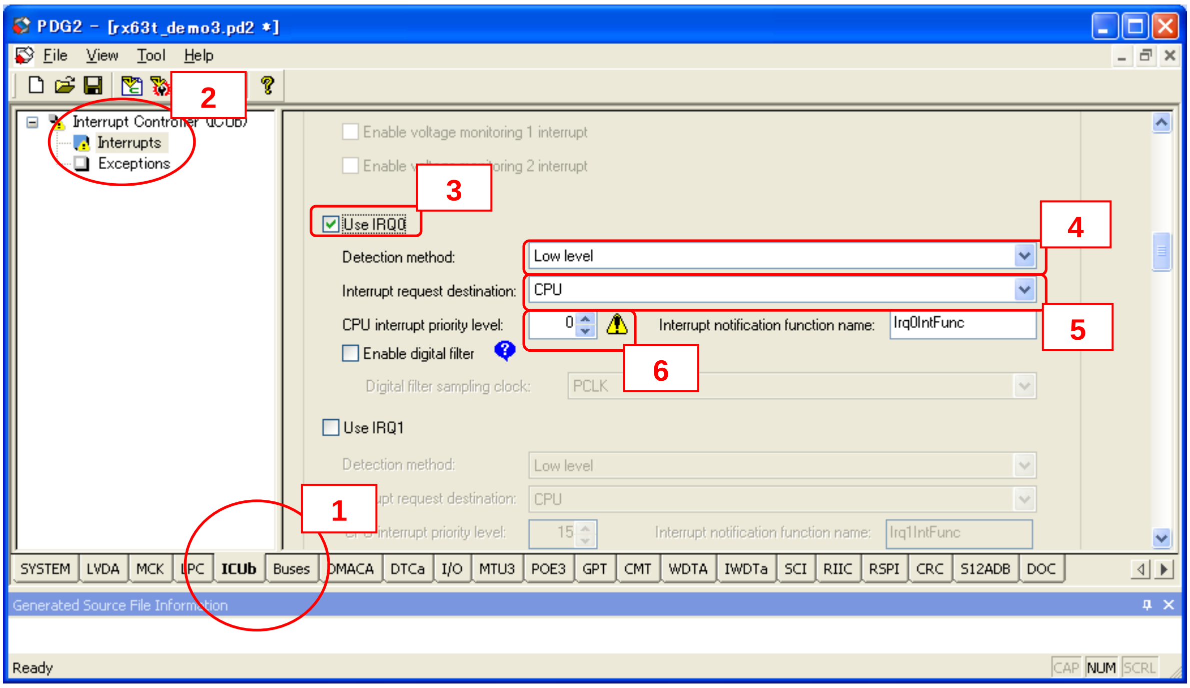
(5) DTCa setting-2 **PDG**

1. Click [Add transfer data] to add the transfer data.
2. Select “IRQ0 (external pin interrupt)” for the activating source.
3. Set “C00” to the transfer data start address.
4. Select “Normal transfer mode” for the transfer mode.
5. Set “1” to the transfer unit size.
6. Set “10” to the transfer count.
7. Set “C10” to the source start address.
8. Select “Increment” for the source address mode.
9. Set “C20” to the destination start address.
10. Select “Increment” for the destination address mode.



(6) ICUb setting **PDG**

1. Select “ICUb” tab to open the ICUb setting window.
2. Click “Interrupts” on the tree view.
3. Check "Use IRQ0".
4. Select “Falling edge” for the detection method of IRQ0.
5. Select “CPU (After activating DTC and data transfer completion)”.
6. CPU interrupt will not be used then set “0” to the CPU interrupt priority level.



(7) Generating source files **PDG**

To generate source files, click  on the tool bar. For details on generating source files, refer to section 4.1.1 (9), Generating source files.

(8) Preparing the HEW project **HEW**

Start the HEW and make RX63N workspace. For details on making HEW project, refer to section 4.1.1 (10), Preparing the HEW project.

(9) Adding the generated source files to the HEW project **PDG**

To add the generated source files to HEW, click  on the tool bar. For details on adding the source files to HEW project, refer to section 4.1.1 (11), Adding the generated source files to the HEW project.

(10) Making the program on HEW **HEW**

By changing the part of “main” function, make the following program on HEW.

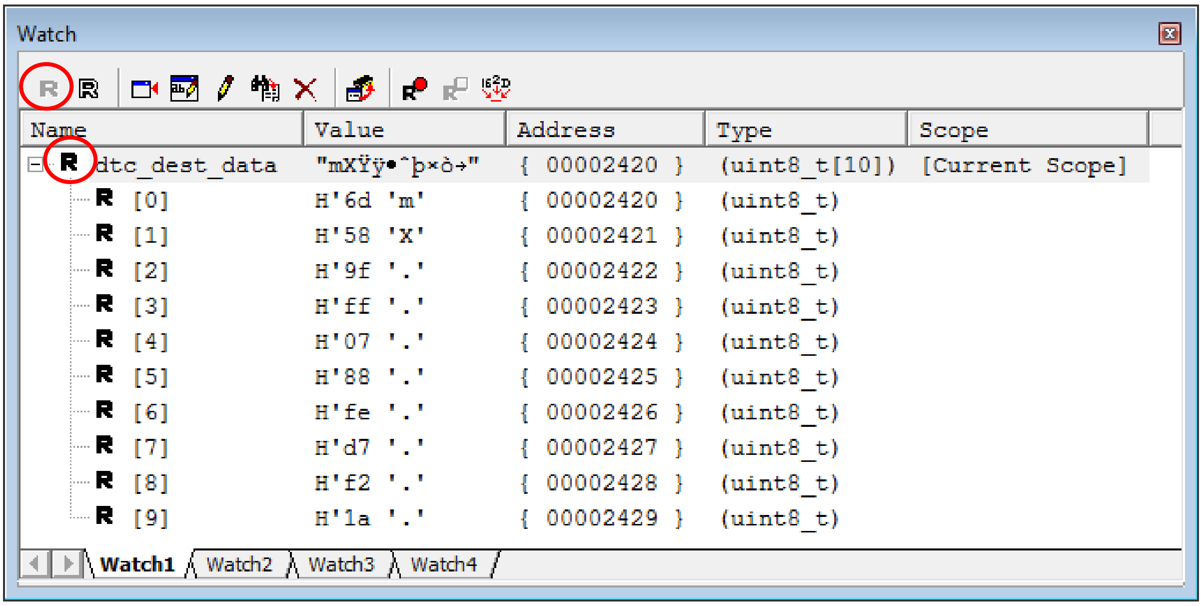
```
//Include "R_PG_<PDG project name>.h"
#include "R_PG_rx63t_demo3.h"

//DTC vector table
#pragma address dtc_vector_table = 0x00000800
uint32_t dtc_vector_table [256];

//DTC transfer data storage area (IRQ0)
#pragma address dtc_transfer_data_IRQ0 = 0x00000C00
uint32_t dtc_transfer_data_IRQ0 [4];

//Transfer source
#pragma address dtc_src_data = 0x00000C10
uint8_t dtc_src_data [10] = "ABCDEFGHIJ";

//Transfer destination
#pragma address dtc_dest_data = 0x00000C20
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;
	}

	R_PG_Clock_WaitSet(0.01);	//Set up the clocks (wait cycle insertion)

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

	// Set up the DTC (transfer data of IRQ0)
	R_PG_DTC_Set_IRQ0();

	R_PG_ExtInterrupt_Set_IRQ0();	// Set up IRQ0

	R_PG_DTC_Activate();	// Make the DTC be ready to the trigger

	while(1);
}
```

(11) 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 4.1.1 (13), connecting to the emulator, building the program and executing.

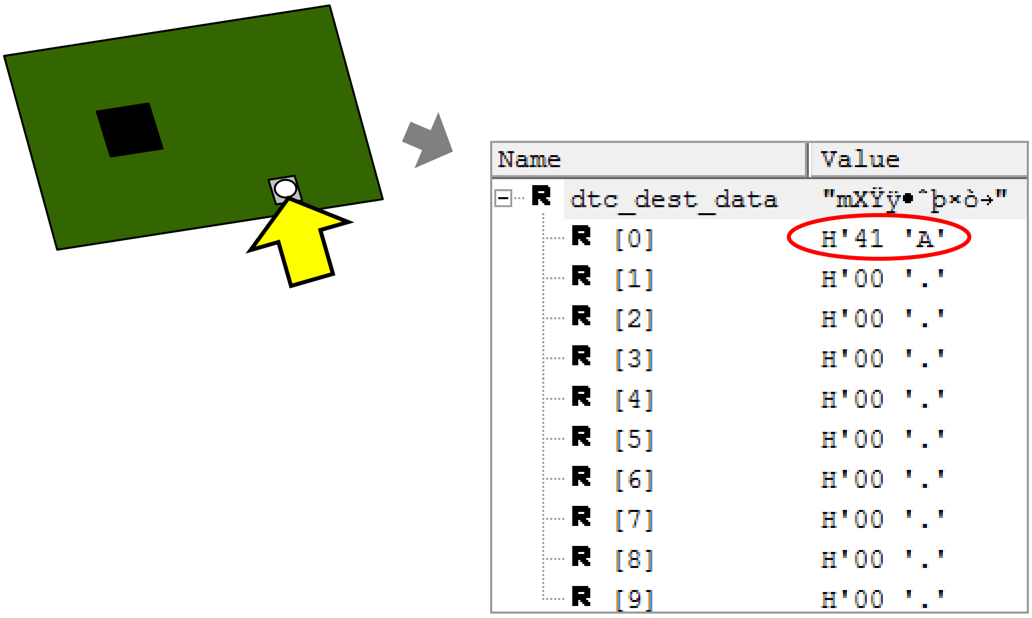
(12) Adding the variable of the transfer destination **HEW**

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.



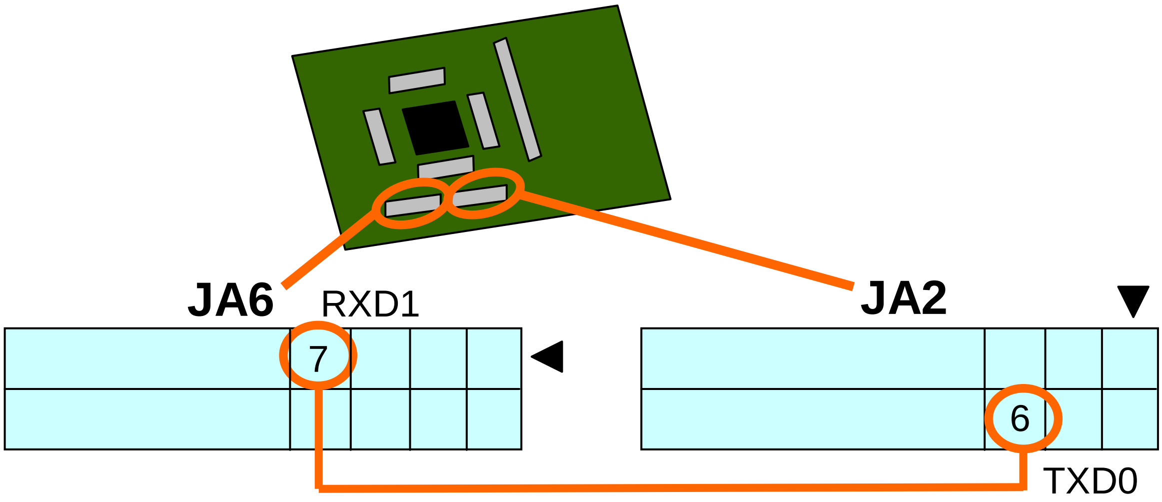
(13) Executing the program and monitoring the result of the transfer **HEW**

Start the execution and push the SW1. The value of “dtc_dest_data” on the watch window will change.



4.4.4 Data transfer between SCIc channels 0 and 1

In this tutorial, SCI channel 0 and 1 will be set up to transfer data in asynchronous mode. Connect the transmission pin of channel 0 (TXD0) and the reception pin of channel 1 (RXD1) on the RSK board as follows.



Note : If there are switches that enables/disables TXD0 and RXD1 on the RSK board, enable it.

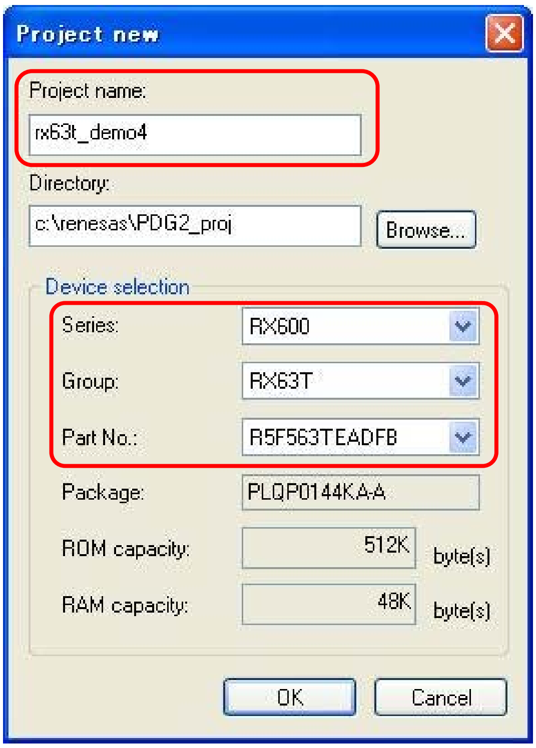
(1) Making the PDG project **PDG**

Make the new PDG project "rx63t_demo4". For details on how to make the new PDG project, refer to section 4.1.1 (1), Making the PDG project.

Set the CPU type as follows.

Series : RX600
Group : RX63T
Part No. : R5F563TEADFB

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



(2) Clock setting **PDG**

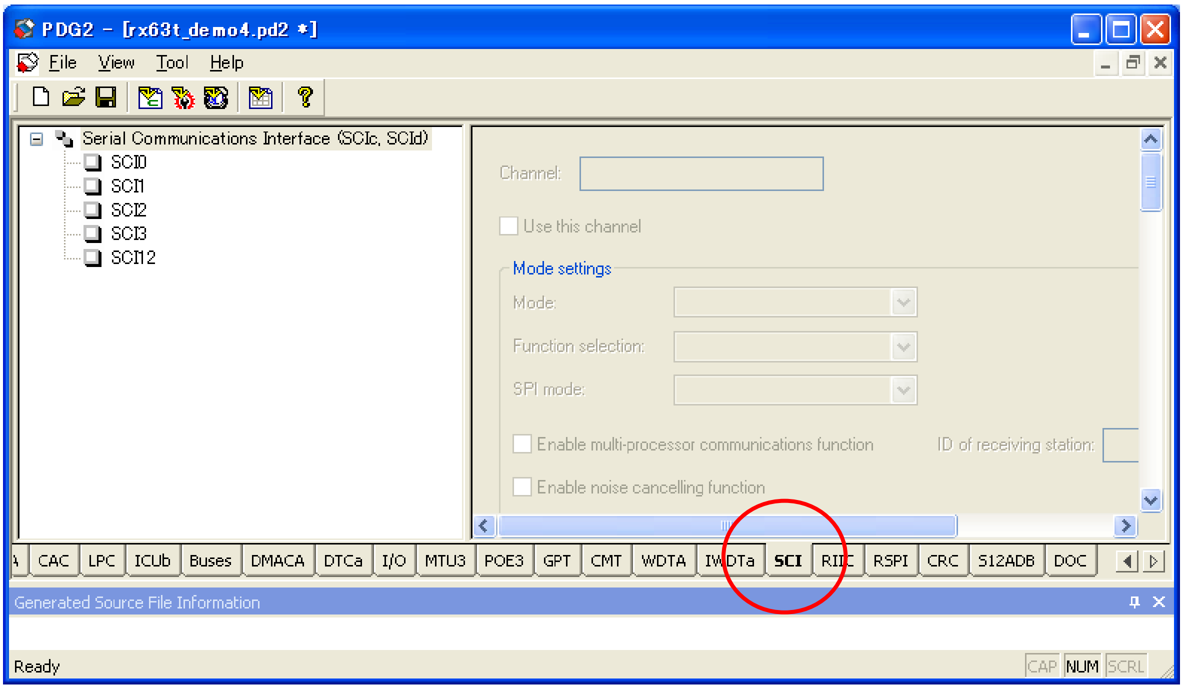
1. The clock setting window opens and the error icons are displayed in the initial state. For icons such as and displayed on window, refer to section 4.1.1 (2), Initial state.
2. For the clock setting, refer to section 4.1.1 (3), Clock setting.

(3) Endian setting **PDG**

For the endian setting, refer to section 3.3, Endian.

(4) SCIc setting **PDG**

Select “SCI” tab to open the SCIc setting window.

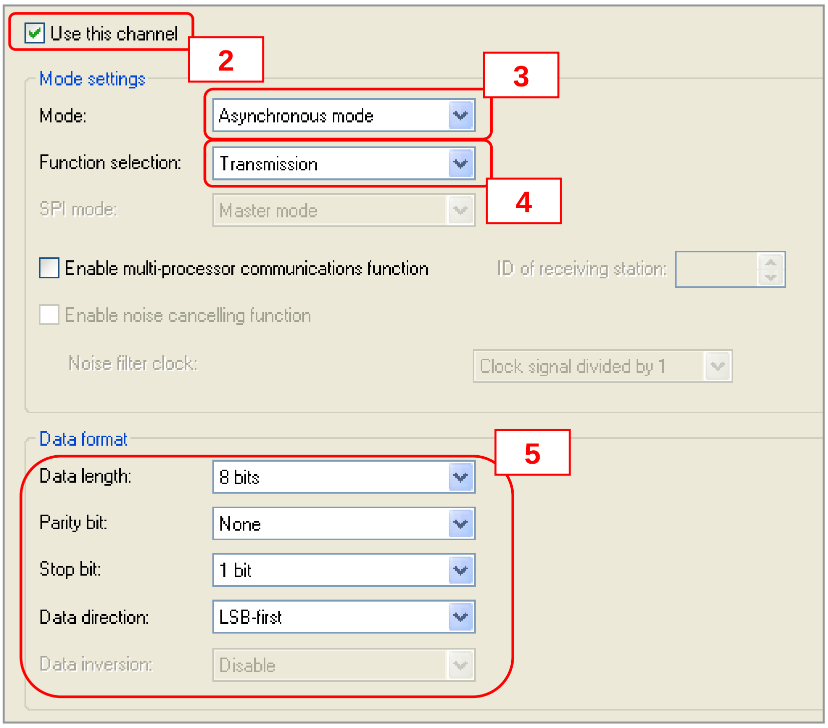


(5) SCI0 (transmitter) setting **PDG**

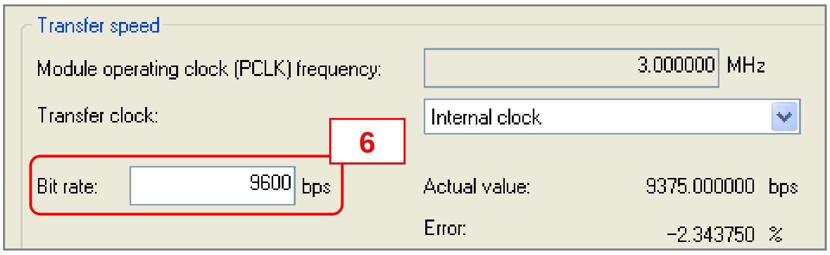
Make the setting for SCI0 as follows.

1. Select SCI0 on the tree view.

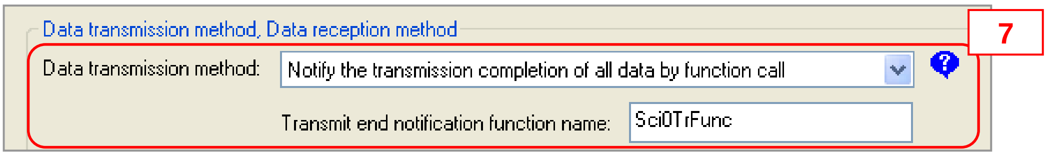
2. Check "Use this channel".
3. Select "Asynchronous mode".
4. Select "Transmission" for the function.
5. Leave the data format settings at the default.



6. Set the bit rate to "9600bps".



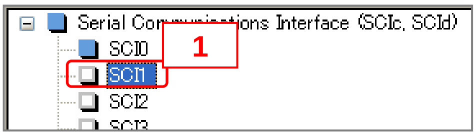
7. Select "Notify the transmission completion of all data by function call" for the data transmission method.



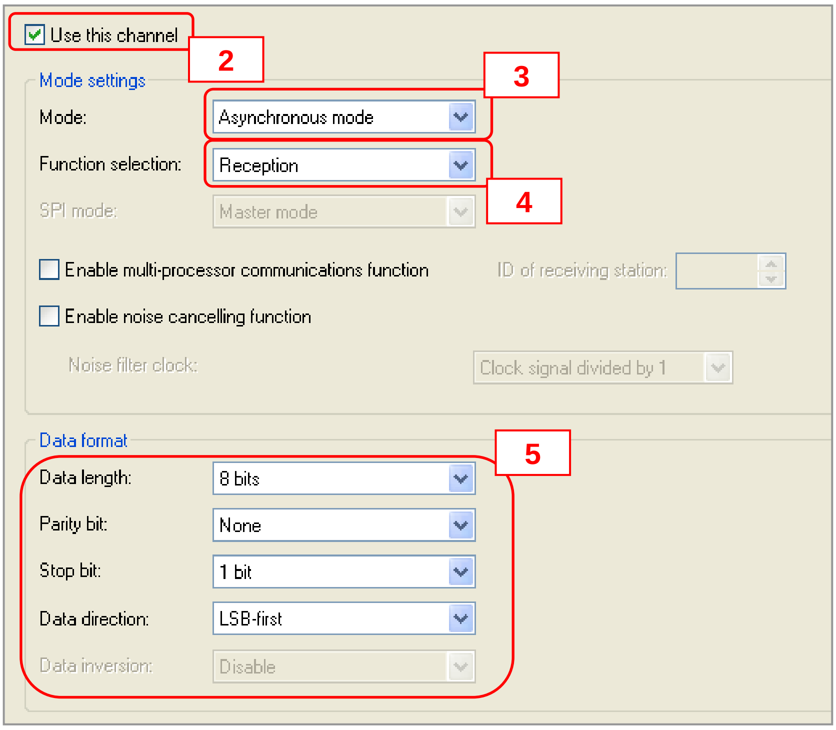
(6) SCI1 (receptor) setting **PDG**

Make the setting for SCI1 as follows.

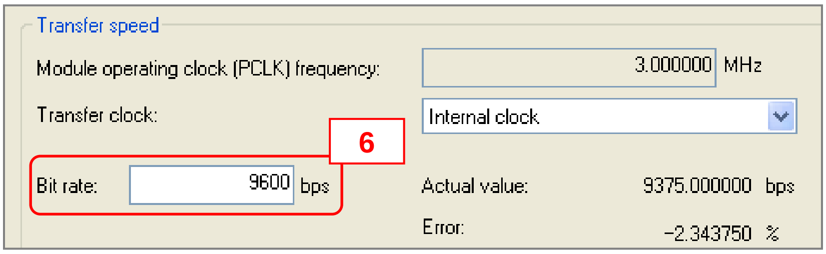
1. Select SCI1 on the tree view.



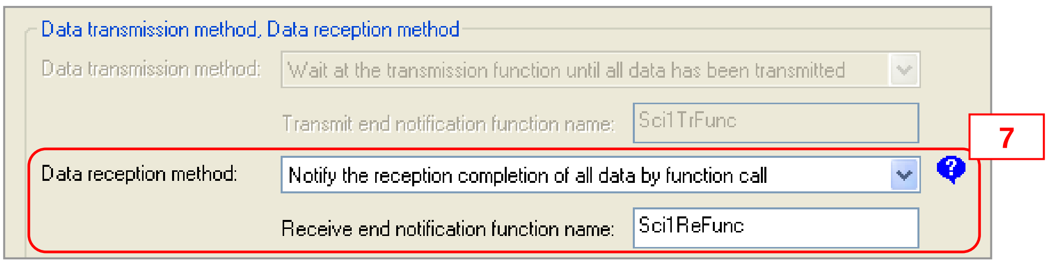
2. Check "Use this channel".
3. Select "Asynchronous mode".
4. Select "Reception" for the function.
5. Leave the data format settings at the default.



6. Set the bit rate to "9600bps".



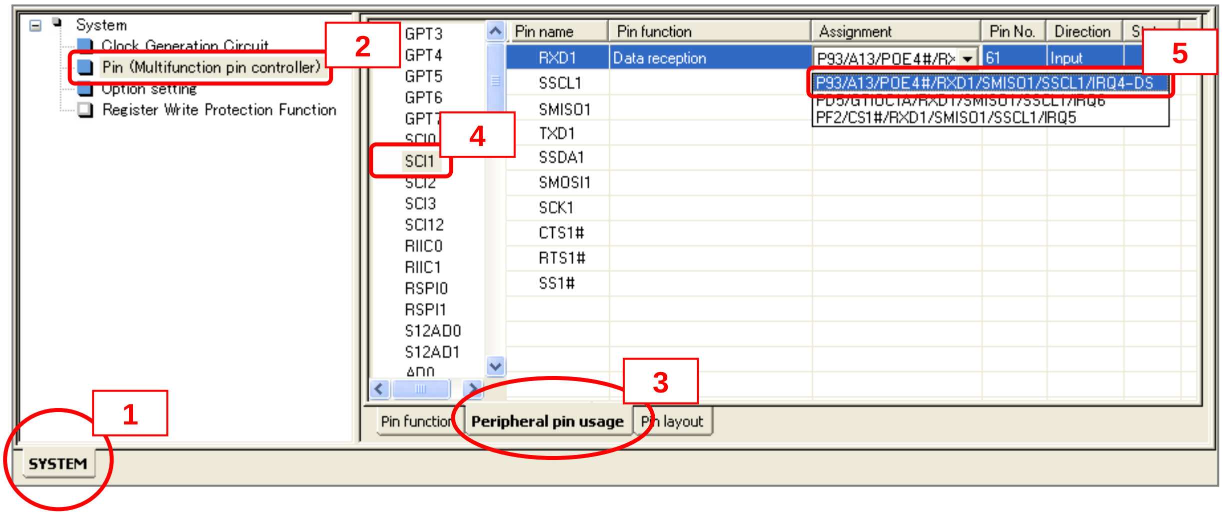
7. Select "Notify the reception completion of all data by function call" for the data reception method.



(7) Pin setting **PDG**

The RXD1 can be assigned to RXD1 (P96) or RXD (PD5) or RXD (PF2). Select the pin function assignment as follows.

1. Select “SYSTEM” tab.
2. Select “Pin(Multifunction pin controller)” on tree view.
3. Select “Perifheral pin usage” tab.
4. Select “SCI1” from the perifheral module list.
5. When the mouse pointer is placed on “assignment” column of RXD1 line, a dropdown button is displayed. Select “PD5/GTIOC1A/RXD1/SMISO1/SSCL1/IRQ6” from the dropdown list.



(8) Generating source files **PDG**

To generate source files, click  on the tool bar. For details on generating source files, refer to section 4.1.1 (9), Generating source files.

(9) Preparing the HEW project **HEW**

Start the HEW and make RX63T workspace. For details on making HEW project, refer to section 4.1.1 (10), Preparing the HEW project.

(10) Adding the generated source files to the HEW project **PDG**

To add the generated source files to HEW, click  on the tool bar. For details on adding the source files to HEW project, refer to section 4.1.1 (11), Adding the generated source files to the HEW project.

(11) Making the program on HEW **HEW**

By changing the part of “main” function, make the following program on HEW.

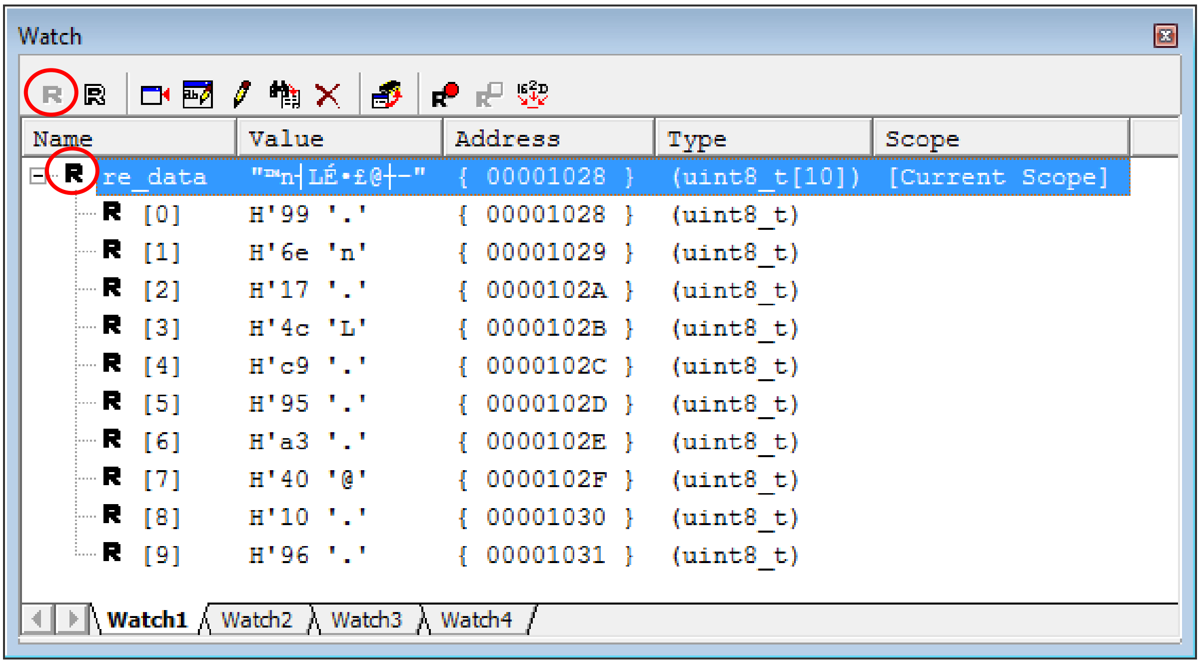
```
//Include "R_PG_<PDG project name>.h"
#include "R_PG_rx63t_demo4.h"

//SCI0 transmission data
uint8_t tr_data[10] = "ABCDEFGHIJ";

//SCI1 reception data storage area
uint8_t re_data[10] = "----------";

void main(void)
{
	//Set up the clocks (wait cycle insertion)
	R_PG_Clock_WaitSet(0.01);

	// Set up the SCI0
	R_PG_SCI_Set_C0();

	// Set up the SCI1
	R_PG_SCI_Set_C1();

	// Start SCI1 reception (number of data : 10)
	R_PG_SCI_StartReceiving_C1( re_data, 10 );

	// Start SCI0 transmission (number of data : 10)
	R_PG_SCI_StartSending_C0( tr_data, 10 );

	while(1);
}

//SCI0 transmission end notification function
void Sci0TrFunc(void)
{
	//Stop SCI0 communication
	R_PG_SCI_StopCommunication_C0();
}

//SCI1 reception end notification function
void Sci1ReFunc(void)
{
	//Stop SCI1 communication
	R_PG_SCI_StopCommunication_C1();
}
```

(12) 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 4.1.1 (13), connecting to the emulator, building the program and executing.

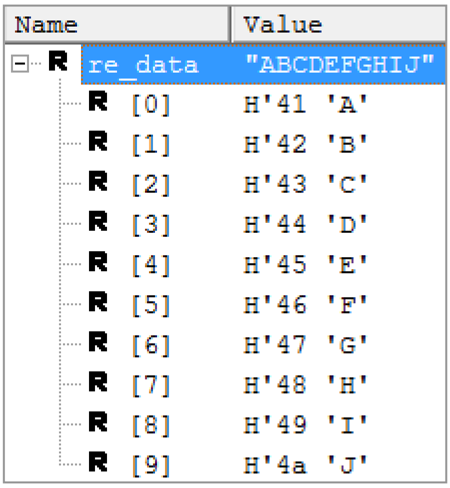
(13) Adding the variable of the reception data **HEW**

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.



(14) 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.



4.5 When the CubeSuite+ is in Use with Renesas starter kit for RX63T(144-pin)

This section introduces the usage of the PDG by giving instructions on how to use the PDG and CubeSuite+ to create a tutorial program that implements the following operations on the Renesas Starter Kit board for the RX63T(144-pin).

・ An LED blinking on Compare Match Timer (CMT) interrupt

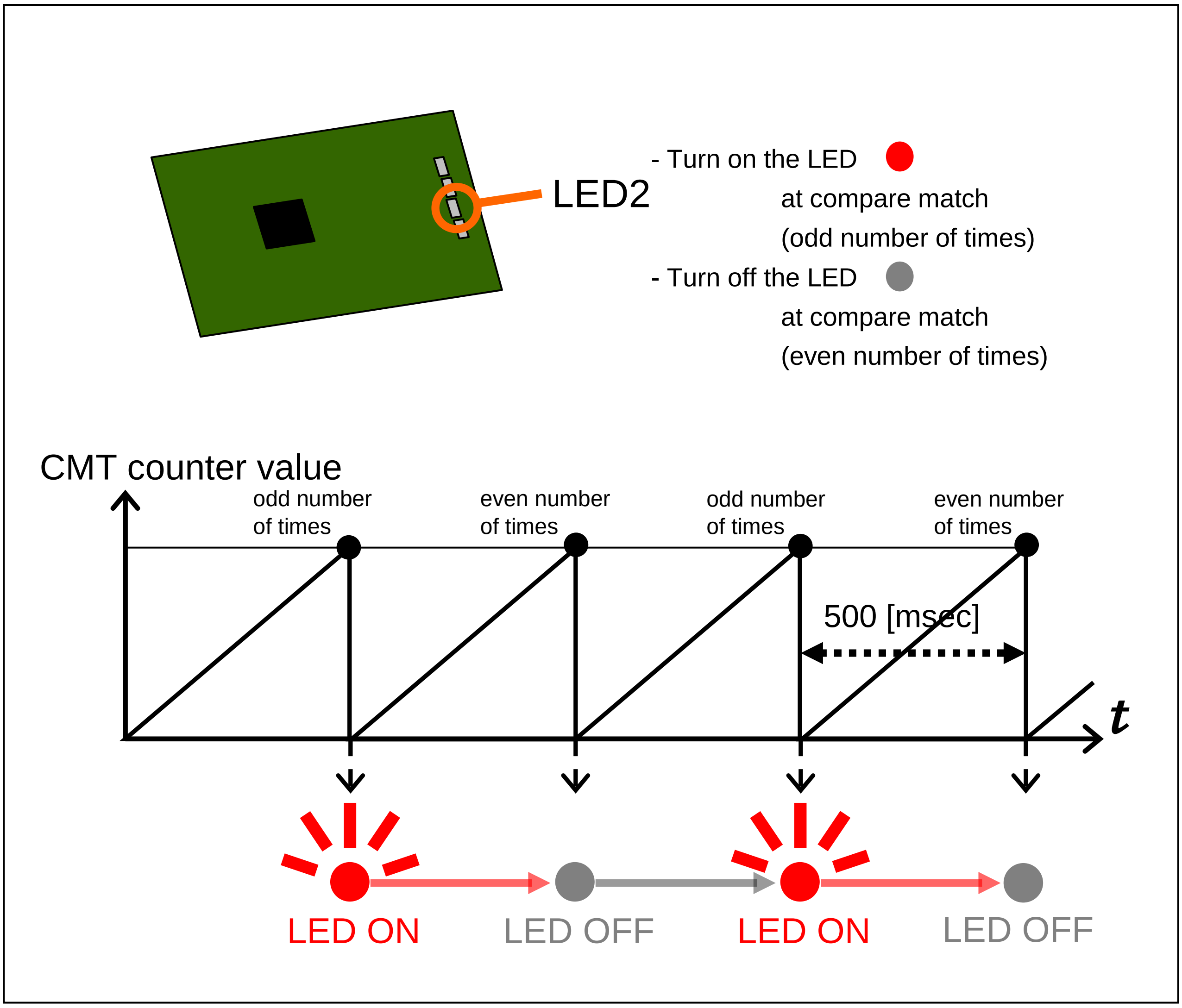
The labels given below respectively indicate operations to take place in the PDG and in the CubeSuite+.

PDG	: Operations in the PDG
CubeSuite+	: Operations in the CubeSuite+

4.5.1 An LED blinking on Compare Match Timer (CMT) interrupt

The LED2 on RSK board is connected to P73. In this tutorial, Compare Match Timer and I/O port will be set up to blink this LED as follows.

Note : If there is a switch that enables/disables P73 on the RSK board, enable it.

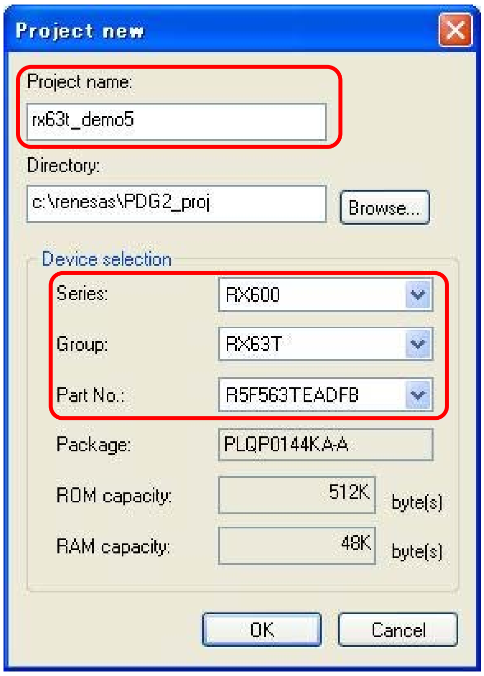


(1) Making the PDG project **PDG**

Make the new PDG project “rx63t_demo5”. For details on how to make the new PDG project, refer to section 4.1.1 (1), Making the PDG project.
Set the CPU type as follows.

Series : RX600
Group : RX63T
Part No. : R5F563TEADFB

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 4.1.1 (2), Initial state.
2. For the clock setting, refer to section 4.1.1 (3), Clock setting.

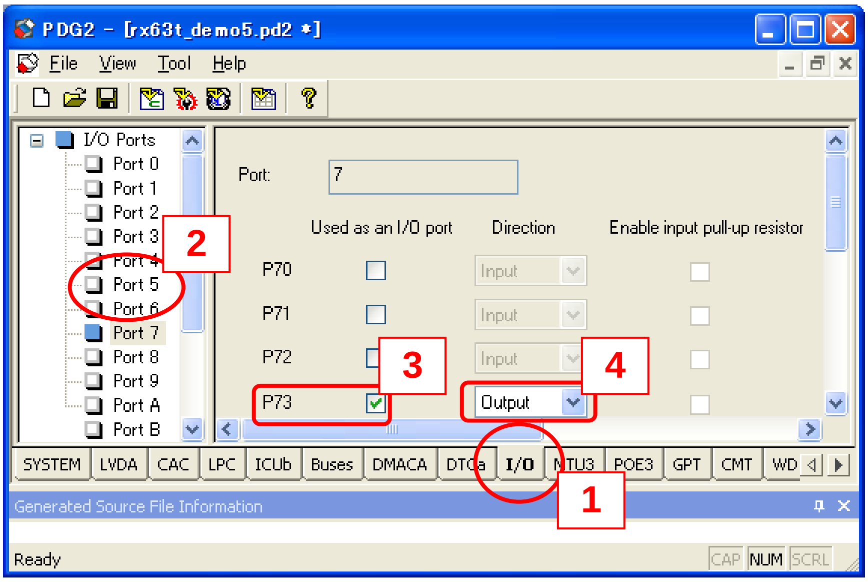
(3) Endian setting **PDG**

For the endian setting, refer to section 3.3, Endian.

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

The LED2 on RSK is connected to P so set P73 to output port.

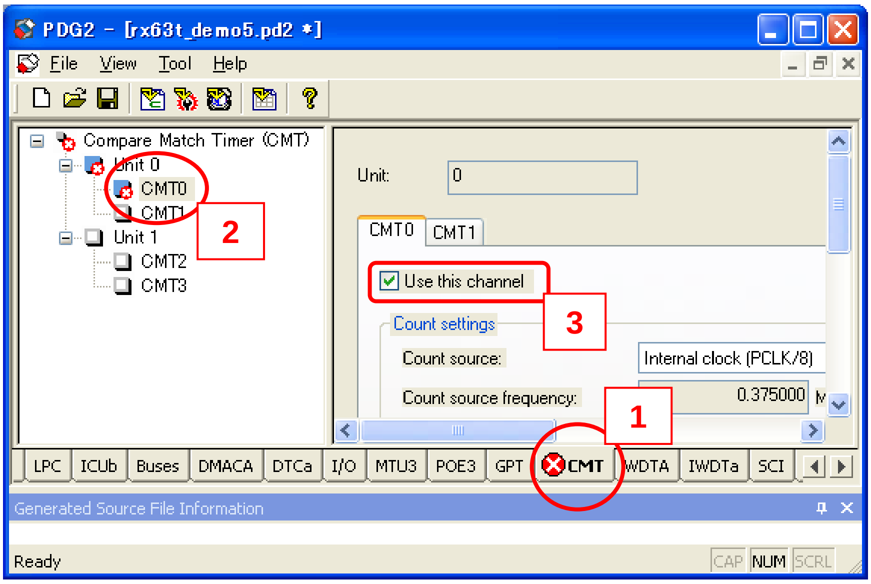
1. Select “I/O” tab
2. Select “Port 7”
3. Check “P73”
4. Select “Output”



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

In this tutorial, CMT0 is used.

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

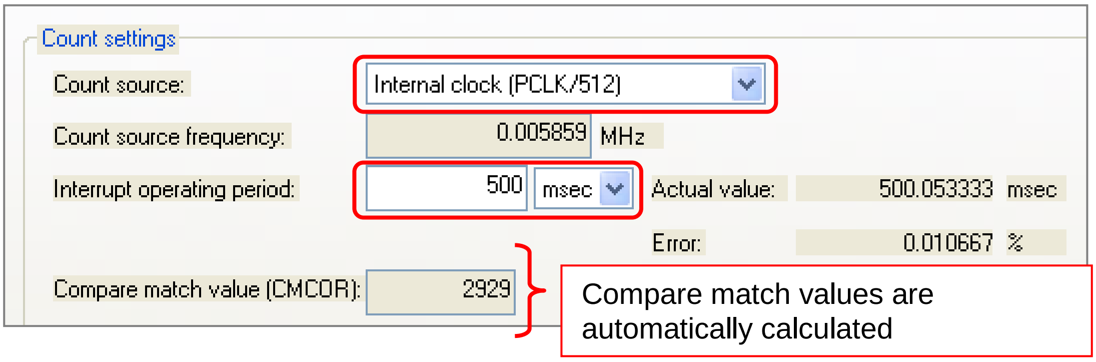


(6) CMT setting-2 **PDG**

Set the other items as follows.

-Count source: Internal clock(PCLK/512)

-Interrupt operating period: 500 msec



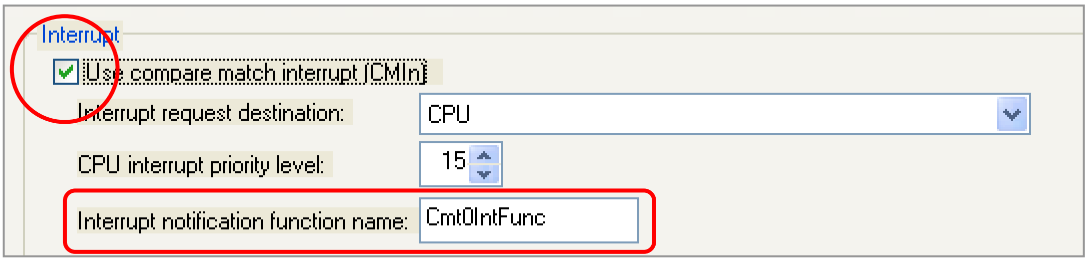
(7) CMT setting-3 **PDG**

Set the interrupt notification functions.

This functions are called when the interrupt occurs.

-Check “Use compare match interrupt (CMIn)”

Notification function name is "Cmt0IntFunc"



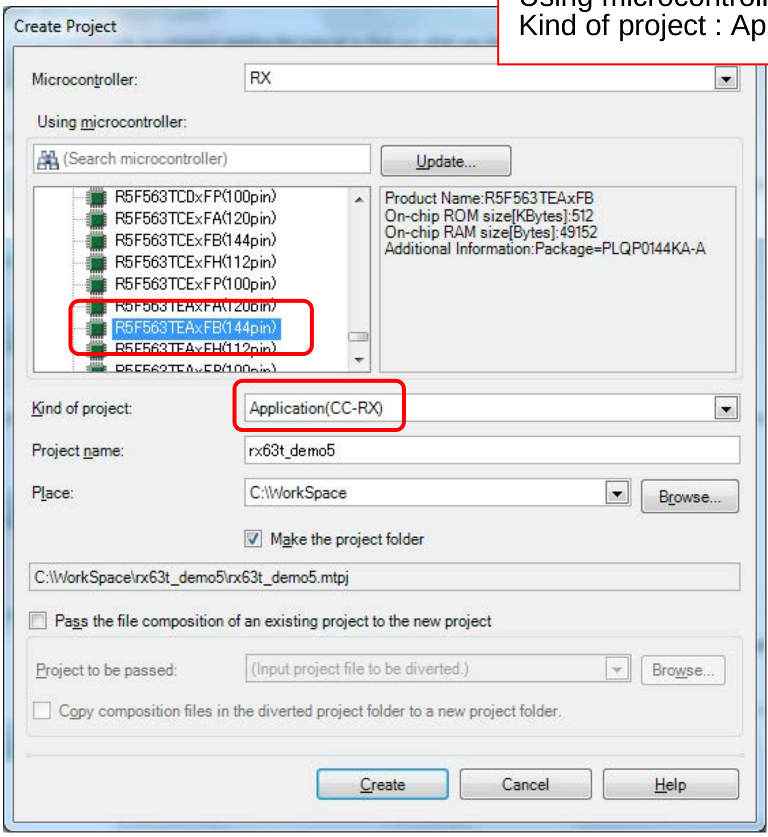
(8) Generating source files **PDG**

To generate source files, click  on the tool bar. For details on generating source files, refer to section 4.1.1 (9), Generating source files.

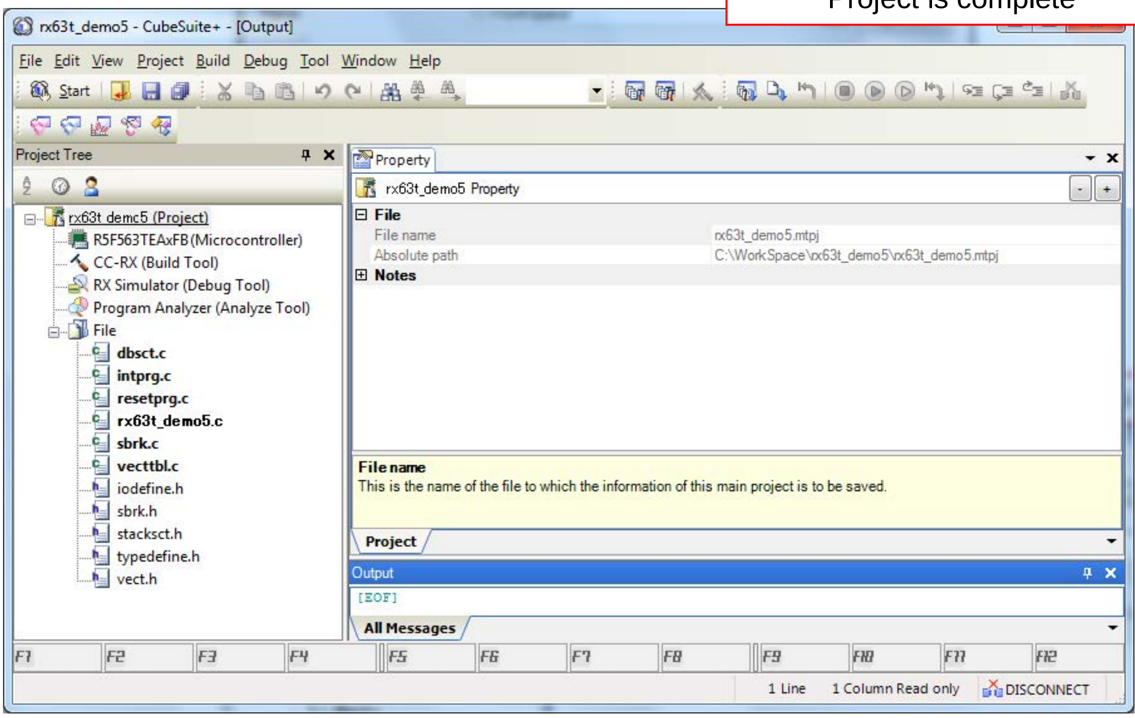
(9) Preparing the CubeSuite+ project **CubeSuite+**

Start the CubeSuite+ and make RX63T workspace.

Using microcontroller : R5F563TEAxFB(144pin)
Kind of project : Application(CC-RX)



Project is complete

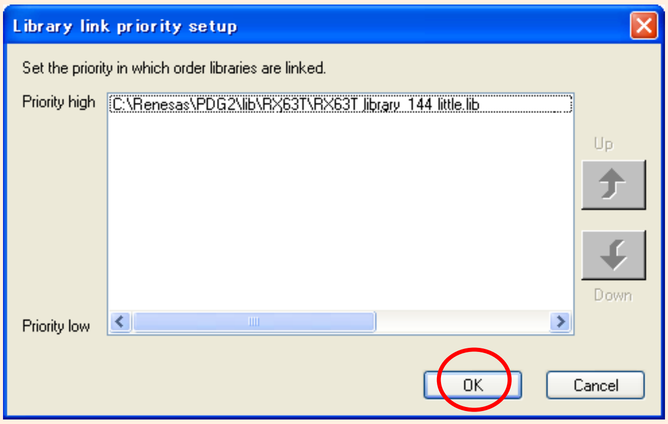


(10) Adding the generated source files to the CubeSuite+ project

PDG

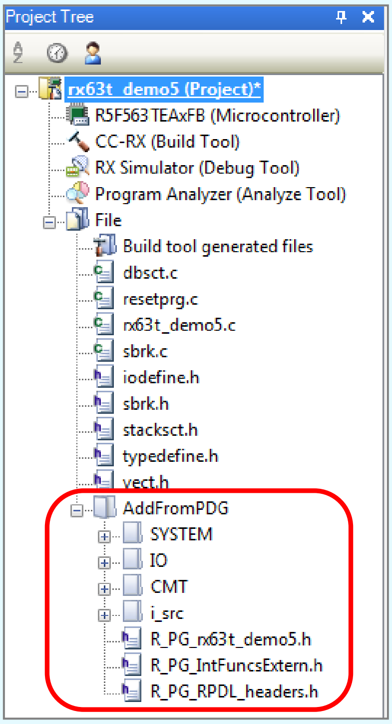
1. To add source files to CubeSuite+, click  on the tool bar.

2. This is a linkage setting of RPDL library.
   When using multiple lib files, linkage order can be set in this dialog box.



CubeSuite+

3. Source files are added to CubeSuite+.
   Added source files are put in "AddFromPDG" category.



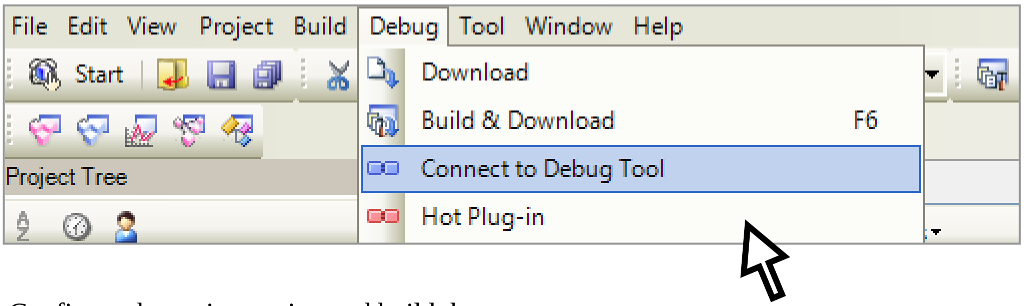
(11) Making the program on CubeSuite+ **CubeSuite+**

By changing the part of “main” function, make the following program on CubeSuite+.

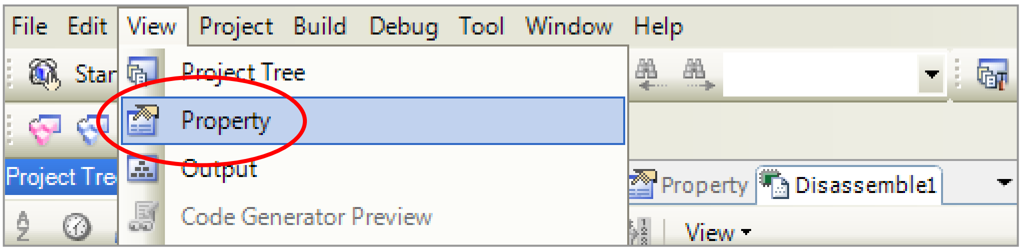
```
//Include "R_PG_<PDG project name>.h"
#include "R_PG_rx63t_demo5.h"

bool led=false;

void main(void)
{
	//Set up the clocks (wait cycle insertion)
	R_PG_Clock_WaitSet(0.01);

	//Set up port P73
	R_PG_IO_PORT_Write_P73(1);	//Initial output value
	R_PG_IO_PORT_Set_P73();

	//Set up the CMT
	R_PG_Timer_Set_CMT_U0_C0();

	//Start the CMT count
	R_PG_Timer_StartCount_CMT_U0_C0();

	while(1);
}

// Compare match interrupt notification function
void Cmt0IntFunc(void)
{
	if( led ){
		//Turn off the LED
		R_PG_IO_PORT_Write_P73(1);
		led = false;
	}
	else{
		//Turn on the LED
		R_PG_IO_PORT_Write_P73(0);
		led = true;
	}
}
```

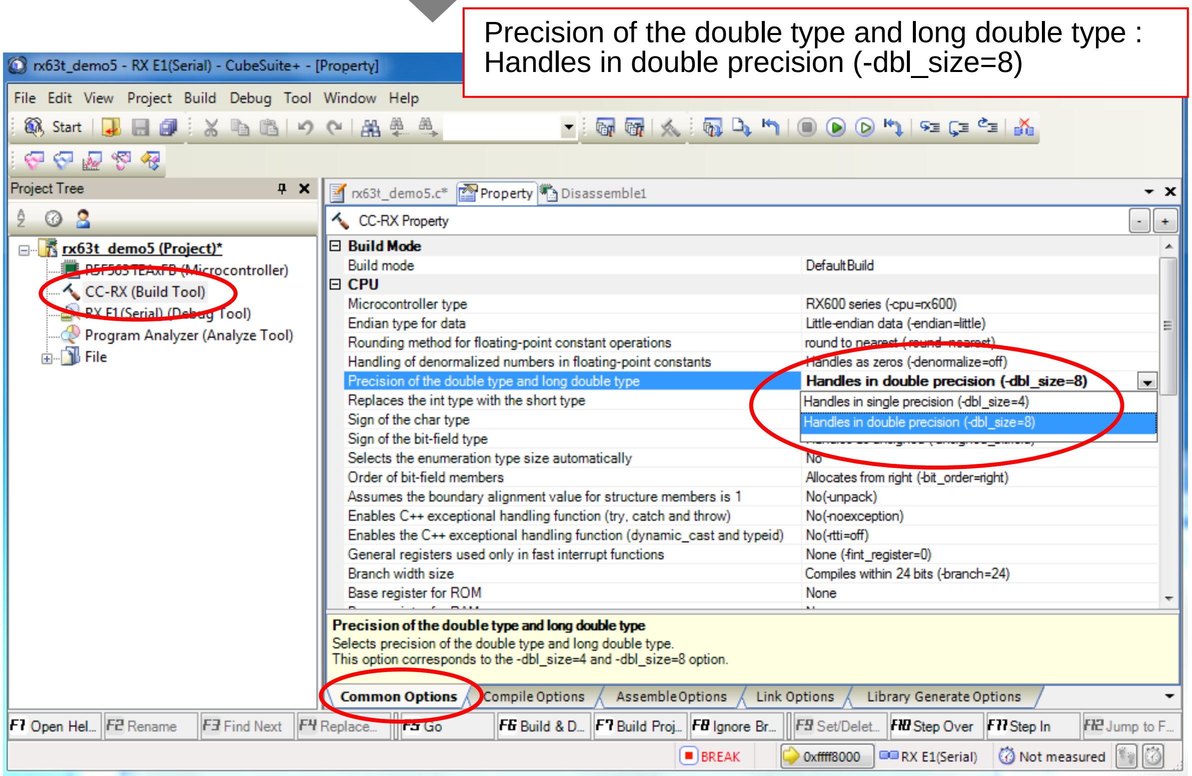
(12) Connecting to the emulator, building the program, downloading and executing **CubeSuite+**

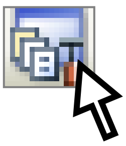
1. Connect to the emulator.



2. Configure the option setting and build the program.

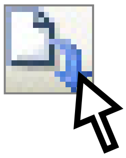






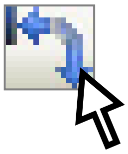
Build button

3. Download the program.

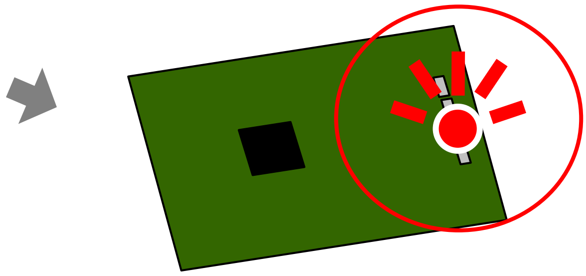


Download button

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



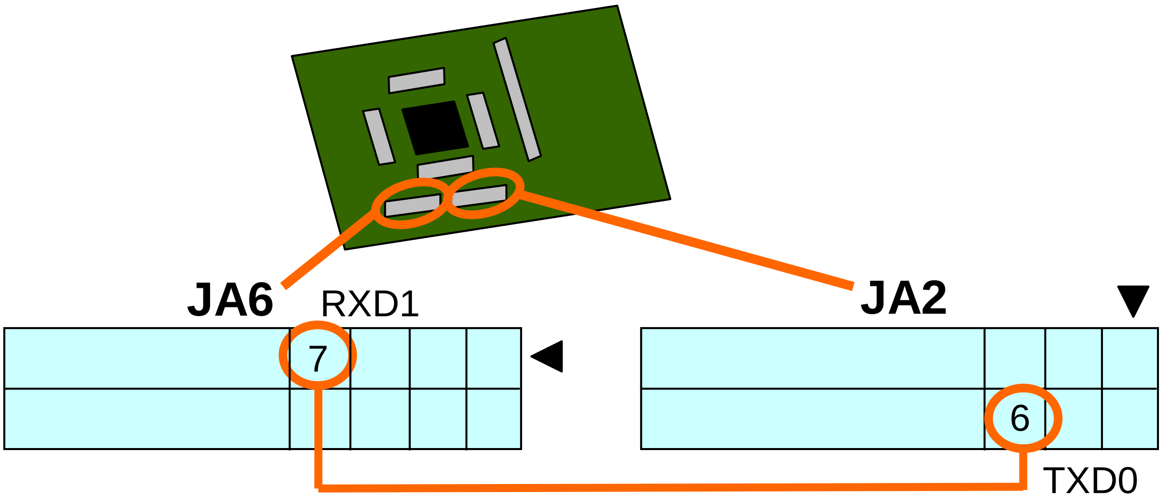
Reset go button



4.6 When the e2 studio is in Use with Renesas starter kit for RX63T(144-pin)

4.6.1 Data transfer between SCIc channels 0 and 1

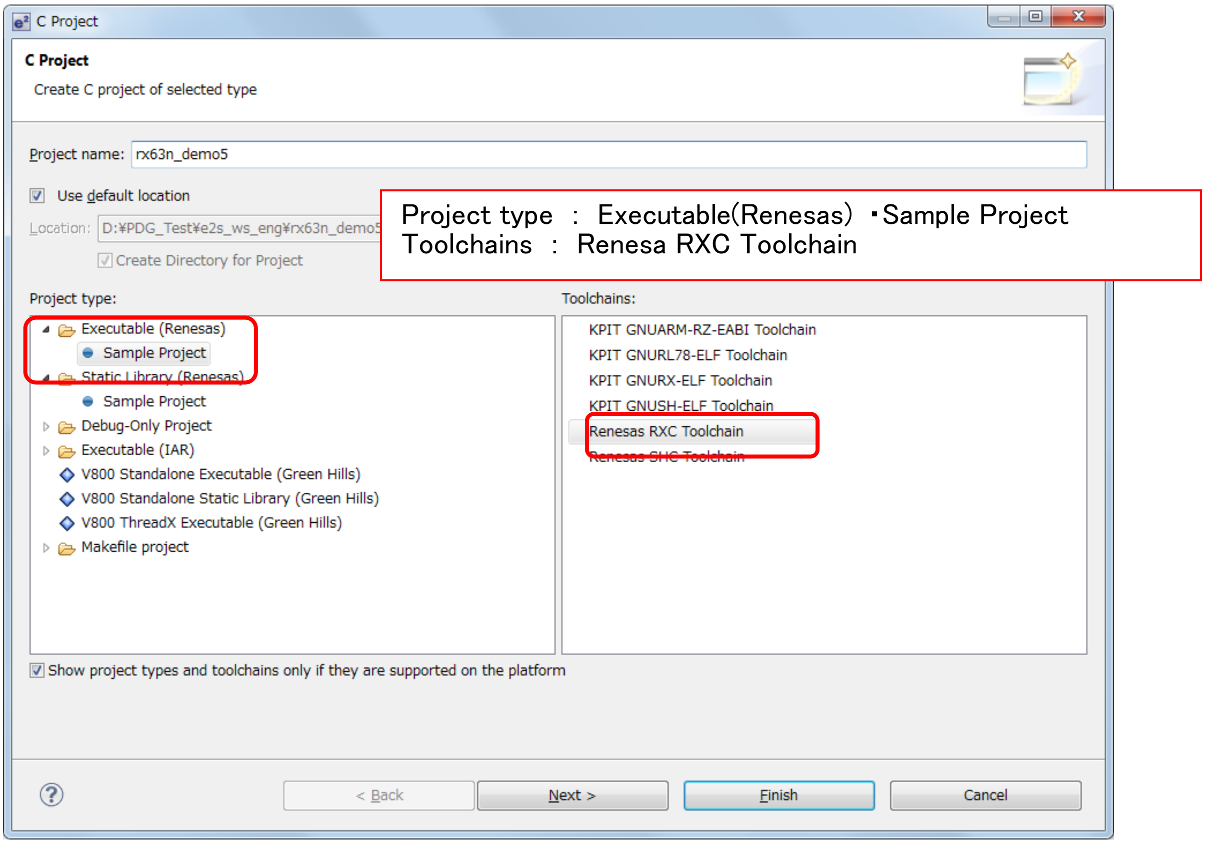
In this tutorial, SCI channel 0 and 1 will be set up to transfer data in asynchronous mode. Connect the transmission pin of channel 0 (TXD0) and the reception pin of channel 1 (RXD1) on the RSK board as follows.

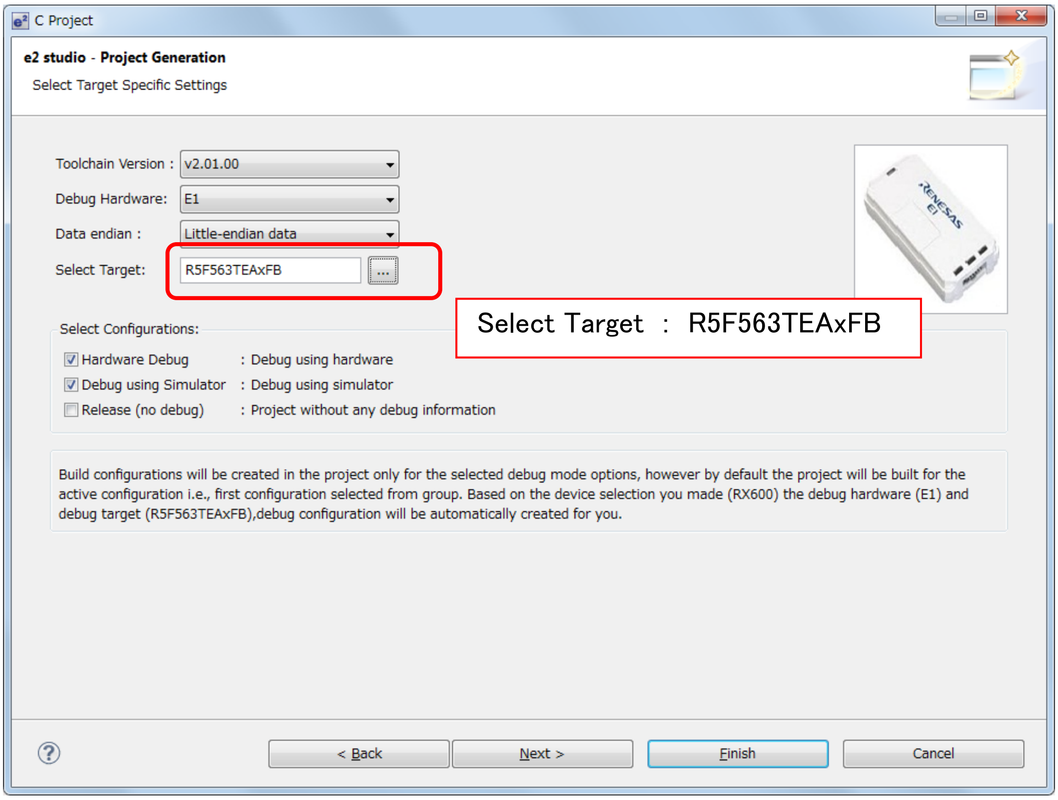


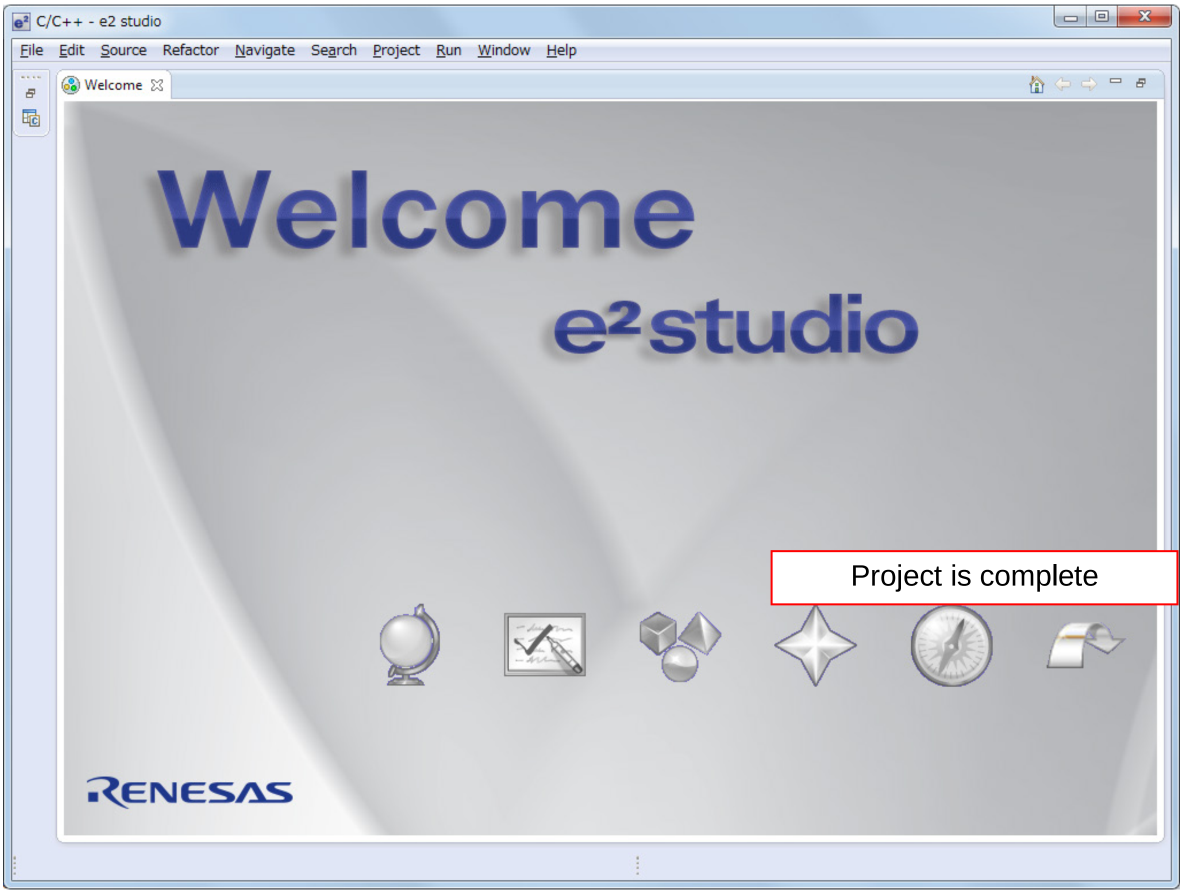
Note : If there are switches that enables/disables TXD0 and RXD1 on the RSK board, enable it.

(1) Preparing the e2 studio project **e2 studio**

Start the e2 studio and make RX63T workspace.




C Project
e2 studio - Project Generation
Select Target Specific Settings
Toolchain Version : v2.01.00
Debug Hardware: E1
Data endian : Little-endian data
Select Target: R5F563TEAxFB
Select Target : R5F563TEAxFB
Select Configurations:
Hardware Debug : Debug using hardware
Debug using Simulator : Debug using simulator
Release (no debug) : Project without any debug information
Build configurations will be created in the project only for the selected debug mode options, however by default the project will be built for the active configuration i.e., first configuration selected from group. Based on the device selection you made (RX600) the debug hardware (E1) and debug target (R5F563TEAxFB),debug configuration will be automatically created for you.
< Back
Next >
Finish
Cancel


C/C++ - e2 studio
File Edit Source Refactor Navigate Search Project Run Window Help
Welcome
Welcome
e²studio
Project is complete
RENESAS

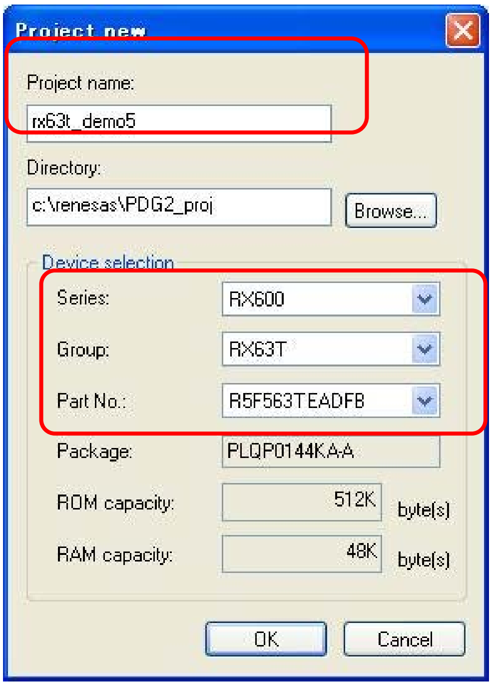
(2) Making the PDG project **PDG**

Make the new PDG project "rx63t_demo4". For details on how to make the new PDG project, refer to section 4.1.1 (1), Making the PDG project.

Set the CPU type as follows.

Series : RX600
Group : RX63T
Part No. : R5F563TEADFB

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



(3) Clock setting **PDG**

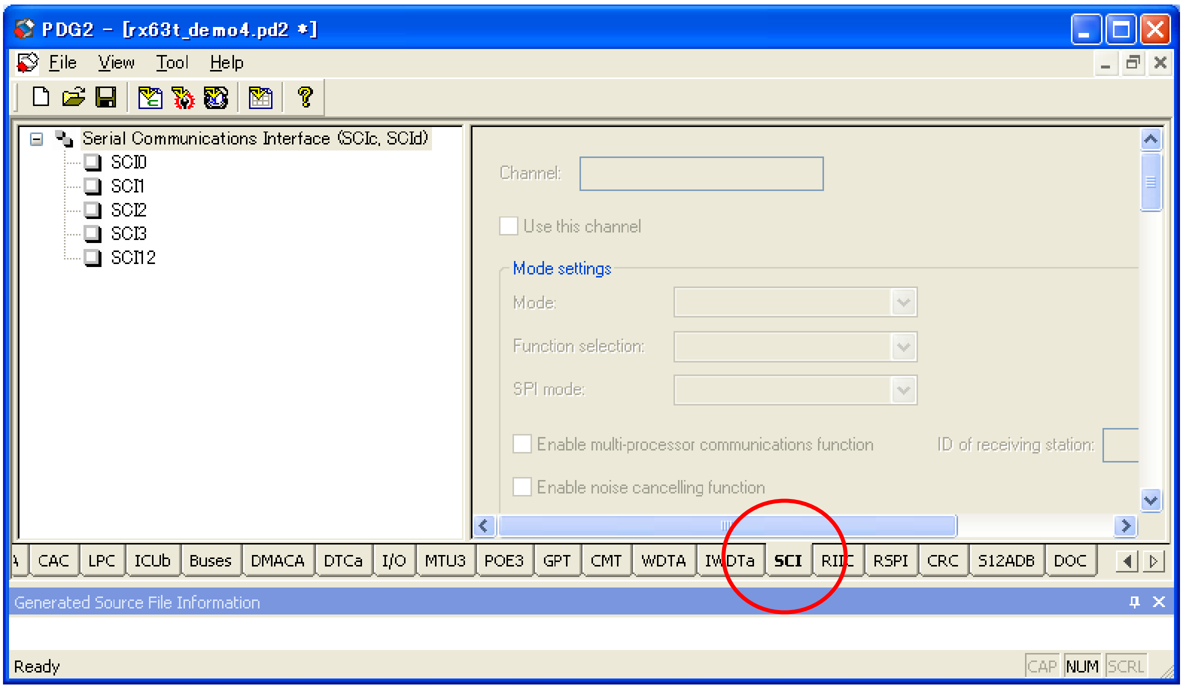
1. The clock setting window opens and the error icons are displayed in the initial state. For icons such as  and  displayed on window, refer to section 4.1.1 (2), Initial state.
2. For the clock setting, refer to section 4.1.1 (3), Clock setting.

(4) Endian setting **PDG**

For the endian setting, refer to section 3.3, Endian.

(5) SCIc setting PDG

Select “SCI” tab to open the SCIc setting window.

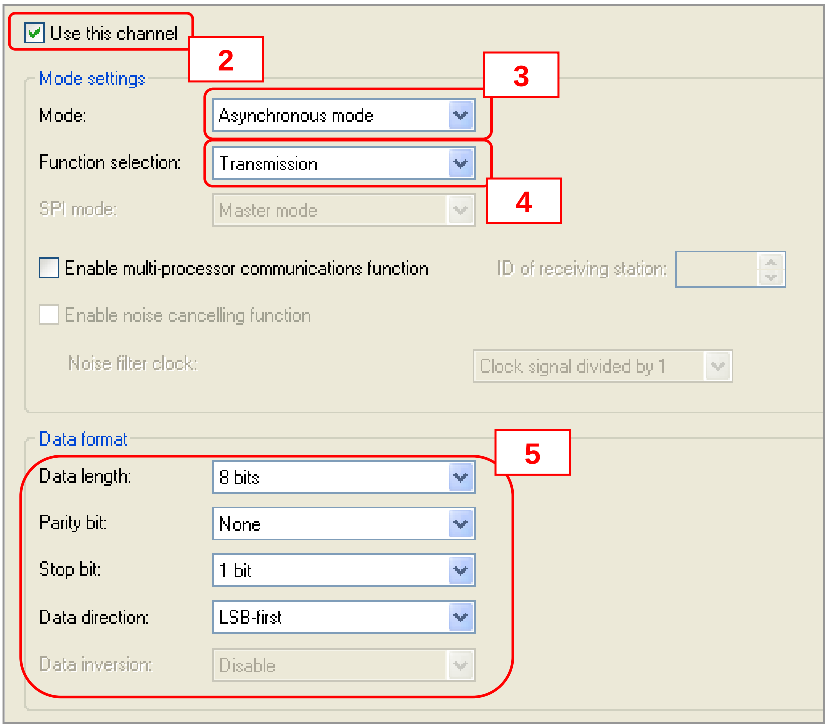


(6) SCI0 (transmitter) setting **PDG**

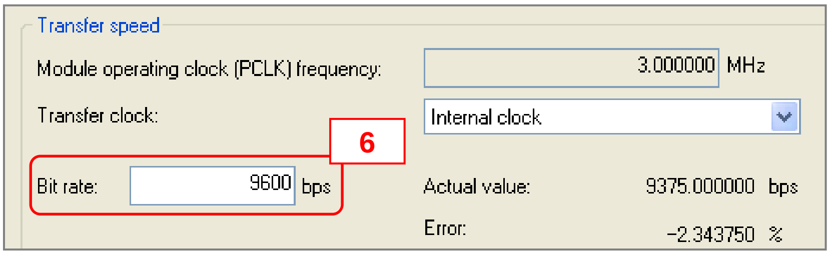
Make the setting for SCI0 as follows.

1. Select SCI0 on the tree view.

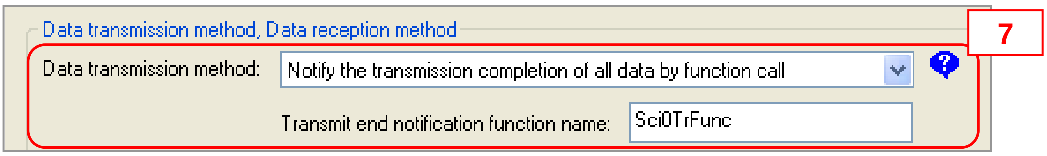
2. Check "Use this channel".
3. Select "Asynchronous mode".
4. Select "Transmission" for the function.
5. Leave the data format settings at the default.



6. Set the bit rate to "9600bps".



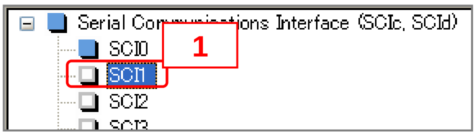
7. Select "Notify the transmission completion of all data by function call" for the data transmission method.



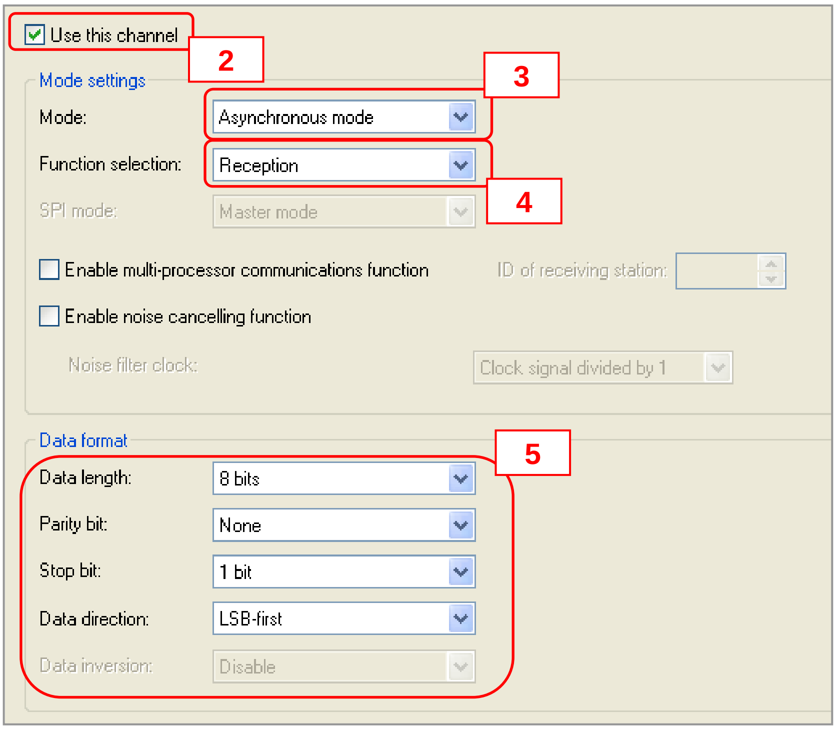
(7) SCI1 (receptor) setting **PDG**

Make the setting for SCI1 as follows.

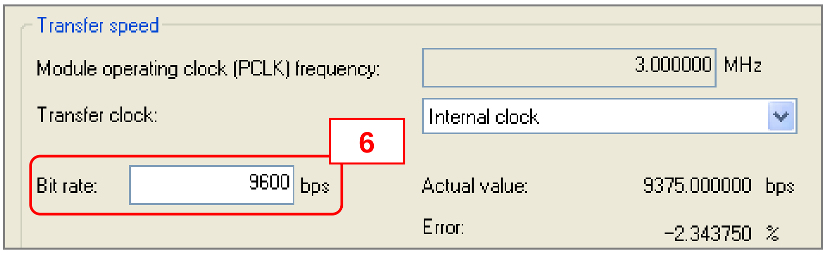
1. Select SCI1 on the tree view.



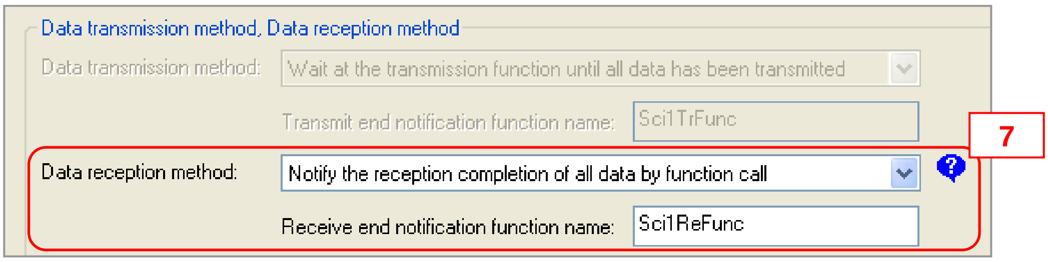
2. Check "Use this channel".
3. Select "Asynchronous mode".
4. Select "Reception" for the function.
5. Leave the data format settings at the default.



6. Set the bit rate to "9600bps".



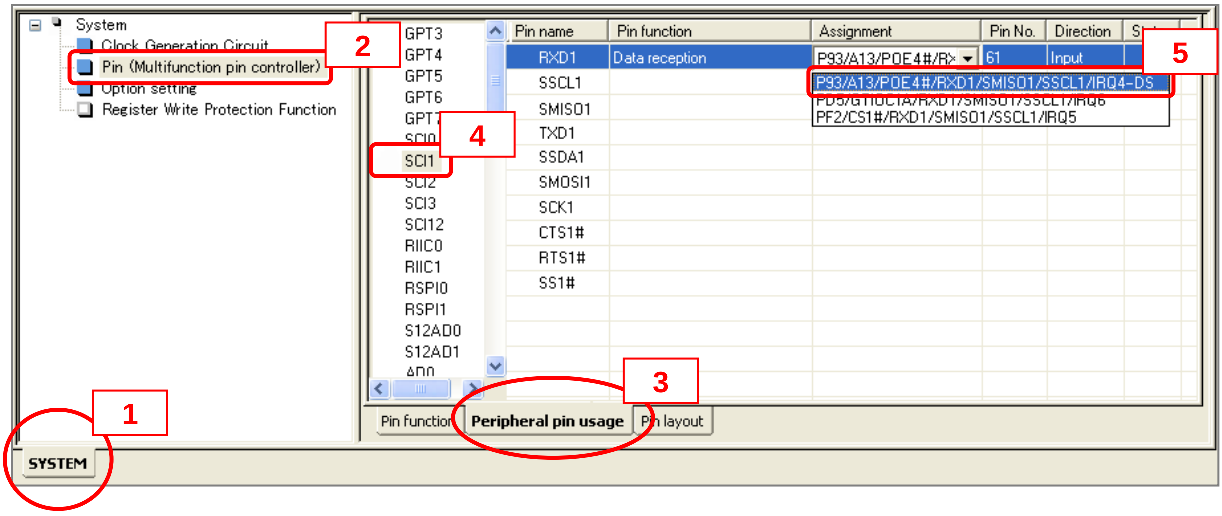
7. Select "Notify the reception completion of all data by function call" for the data reception method.



(8) Pin setting **PDG**

The RXD1 can be assigned to RXD1 (P96) or RXD (PD5) or RXD (PF2). Select the pin function assignment as follows.

1. Select “SYSTEM” tab.
2. Select “Pin(Multifunction pin controller)” on tree view.
3. Select “Perifheral pin usage” tab.
4. Select “SCI1” from the perifheral module list.
5. When the mouse pointer is placed on “assignment” column of RXD1 line, a dropdown button is displayed. Select “PD5/GTIOC1A/RXD1/SMISO1/SSCL1/IRQ6” from the dropdown list.



(9) Generating source files **PDG**

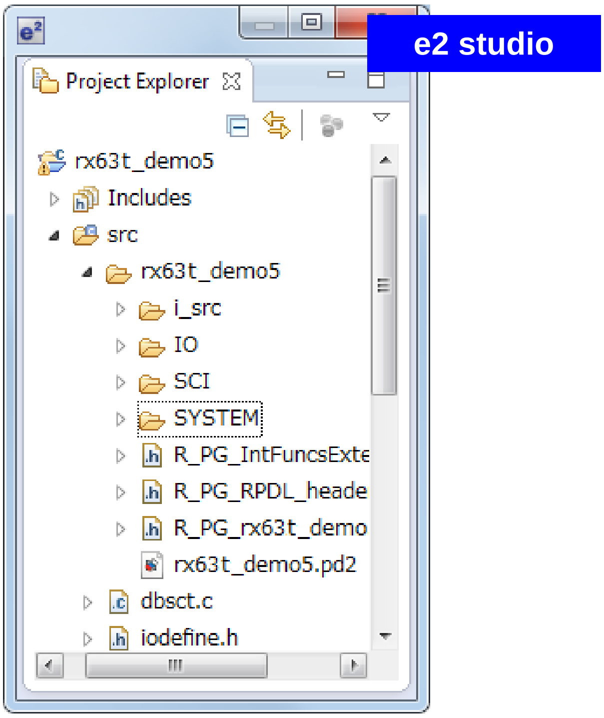
To generate source files, click  on the tool bar. For details on generating source files, refer to section 4.1.1 (9), Generating source files.

(10) Adding the generated source files to the e2 studio project **PDG**

To set a build property of e2 studio , click on the tool bar. A project is established besides the registration of afile. Please refer to "6 About registration to IDE of a generation file" about setting of a project.

A file is added to the project of e2 studio.

An added file is registered by a folder image of a generation source of Peripheral Driver Generator.

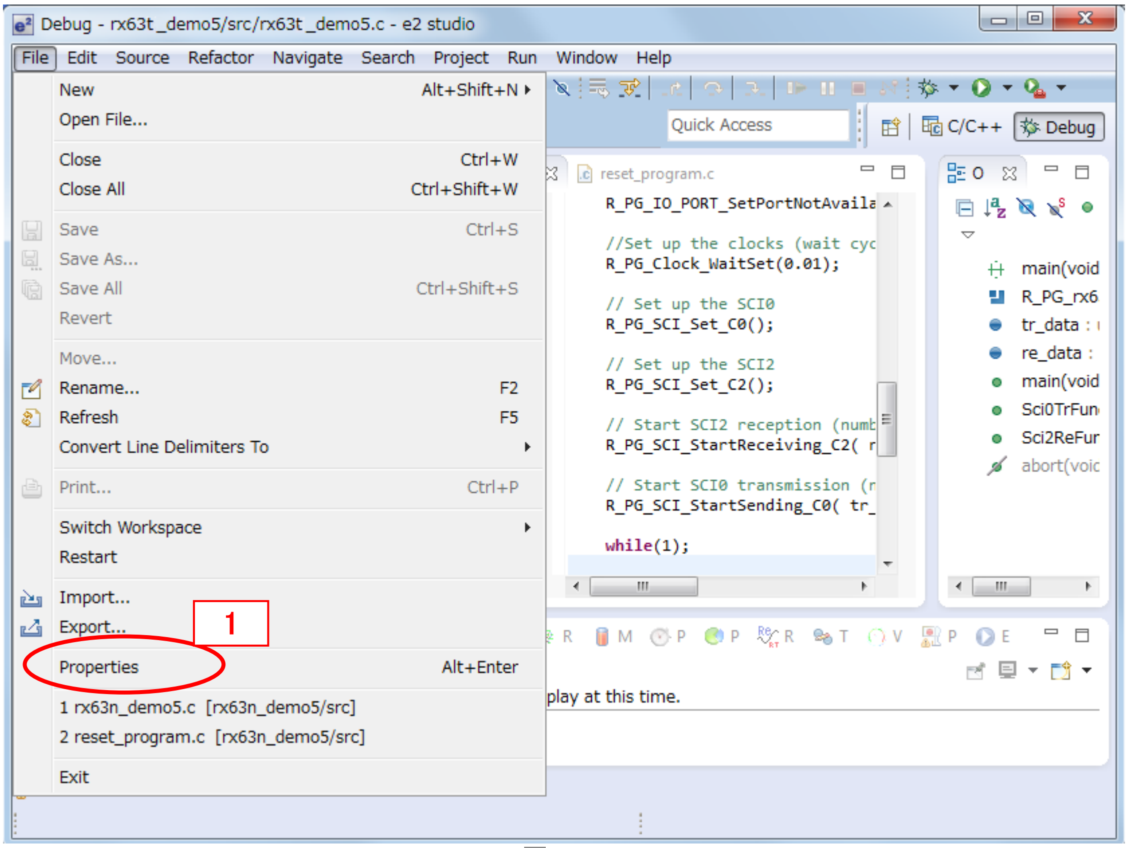


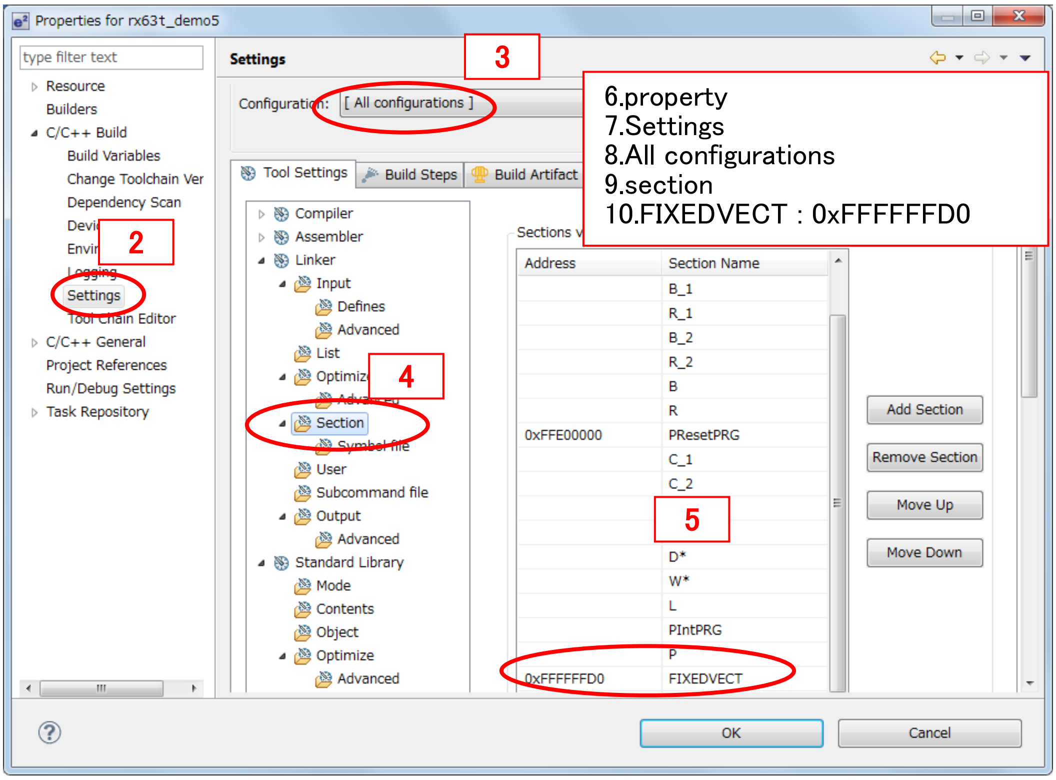
(11) Making the program on e2 studio **e2 studio**

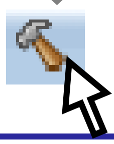
By changing the part of “main” function, make the following program on e2 studio.

```
//Include "R_PG_<PDG project name>.h"
#include "R_PG_rx63t_demo5.h"

//SCI0 transmission data
uint8_t tr_data[10] = "ABCDEFGHIJ";

//SCI1 reception data storage area
uint8_t re_data[10] = "----------";

void main(void)
{
	//Set up the clocks (wait cycle insertion)
	R_PG_Clock_WaitSet(0.01);

	// Set up the SCI0
	R_PG_SCI_Set_C0();

	// Set up the SCI1
	R_PG_SCI_Set_C1();

	// Start SCI1 reception (number of data : 10)
	R_PG_SCI_StartReceiving_C1( re_data, 10 );

	// Start SCI0 transmission (number of data : 10)
	R_PG_SCI_StartSending_C0( tr_data, 10 );

	while(1);
}

//SCI0 transmission end notification function
void Sci0TrFunc(void)
{
	//Stop SCI0 communication
	R_PG_SCI_StopCommunication_C0();
}

//SCI1 reception end notification function
void Sci1ReFunc(void)
{
	//Stop SCI1 communication
	R_PG_SCI_StopCommunication_C1();
}
```

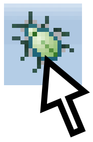
(12) Connecting to the emulator, building the program and downloading **e2 studio**





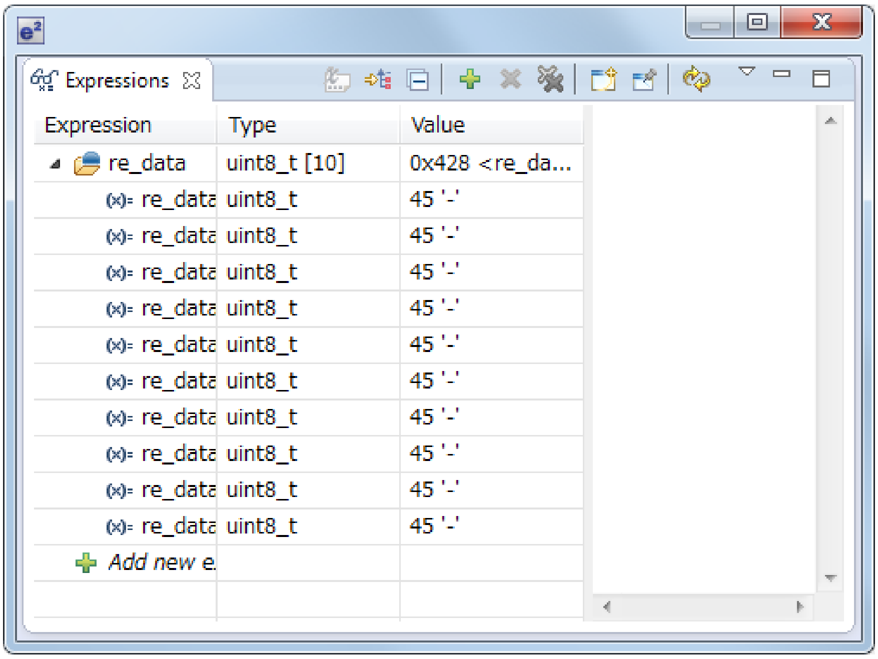


4.download a program.



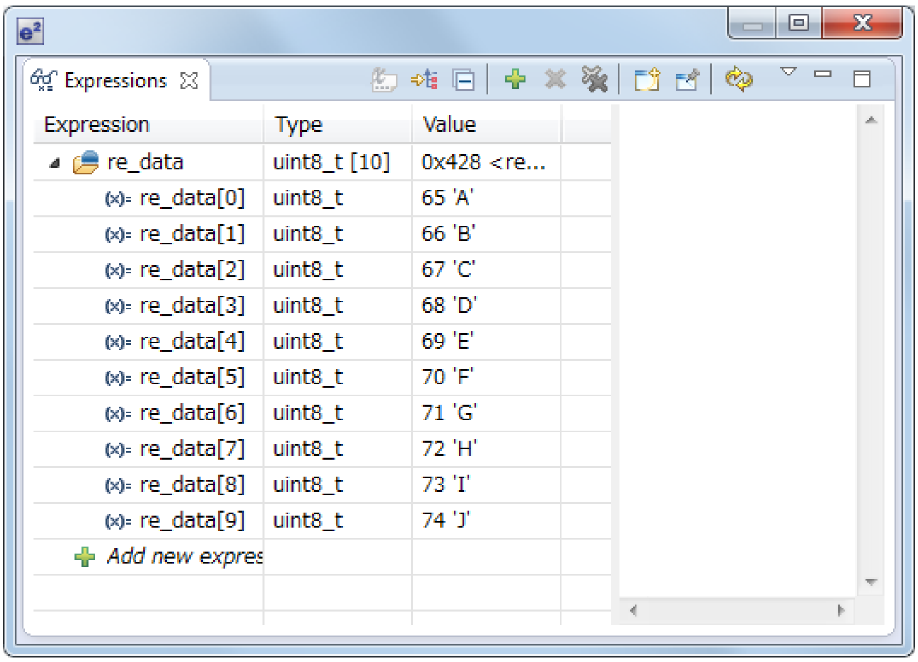
(13) Adding the variable of the reception data **e2 studio**

Open the Expressions window and add the variable "re_data".



(14) Executing the program and monitoring the result of the transfer **e2 studio**

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 RX63T.

Table 5.1 Generated Functions for the RX63T

Clock-generation circuit

Generated Function	Description
R_PG_Clock_Set	Set up the clocks
R_PG_Clock_WaitSet	Set up the clocks (wait cycle insertion)
R_PG_Clock_Start_MAIN	Start the main clock oscillator
R_PG_Clock_Stop_MAIN	Stop the main clock oscillator
R_PG_Clock_Enable_MAIN_ForcedOscillation	Enable the main clock forced oscillation
R_PG_Clock_Disable_MAIN_ForcedOscillation	Disable the main clock forced oscillation
R_PG_Clock_Start_LOCO	Start the low-speed on-chip oscillator (LOCO)
R_PG_Clock_Stop_LOCO	Stop the low-speed on-chip oscillator (LOCO)
R_PG_Clock_Start_PLL	Start the PLL circuit
R_PG_Clock_Stop_PLL	Stop the PLL circuit
R_PG_Clock_Enable_BCLK_PinOutput	Enable BCLK pin output
R_PG_Clock_Disable_BCLK_PinOutput	Disable BCLK pin output
R_PG_Clock_Enable_MAIN_StopDetection	Enable the main clock oscillation stop detection function
R_PG_Clock_Disable_MAIN_StopDetection	Disable the main clock oscillation stop detection function
R_PG_Clock_GetFlag_MAIN_StopDetection	Acquire the main clock oscillation stop detection flag
R_PG_Clock_ClearFlag_MAIN_StopDetection	Clear the main clock oscillation stop detection flag
R_PG_Clock_GetSelectedClockSource	Acquire the current internal clock source
R_PG_Clock_GetClocksStatus	Acquire the status of the clocks

Voltage Detection Circuit (LVDA)

Generated Function	Description
R_PG_LVD_Set	Set up the voltage detection circuit (Voltage-monitoring 1 and 2)
R_PG_LVD_GetStatus	Get the status flag of Voltage Detection Circuit
R_PG_LVD_ClearDetectionFlag_LVD<*Voltage Detection Circuit number*>	Clear Voltage Monitoring n Voltage Change Detection Flag n: 1 or 2
R_PG_LVD_Disable_LVD<*Voltage Detection Circuit number*>	Disable Voltage Monitoring n n: 1 or 2

Clock Frequency Accuracy Measurement Circuit (CAC)

Generated Function	Description
R_PG_CAC_Set	Set up the CAC and start the measurement
R_PG_CAC_ClearFlag_FrequencyError	Clear the frequency error flag
R_PG_CAC_ClearFlag_MeasurementEnd	Clear the measurement end flag
R_PG_CAC_ClearFlag_OverFlow	Crear the overflow flag
R_PG_CAC_StartMeasurement	Start the measurement

R_PG_CAC_StopMeasurement	Stop the measurement
R_PG_CAC_GetStatusFlags	Acquire the CAC status flags
R_PG_CAC_GetCounterBufferRegister	Acquire the counter buffer register (CACNTBR) value
R_PG_CAC_StopModule	Shut down the CAC

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 value of the LVD detection flags
R_PG_LPC_GetDeepSoftwareStandbyResetFlag	Acquire the value of the deep software standby reset flag
R_PG_LPC_GetStatus	Get the status of the low power consumption functions
R_PG_LPC_WriteBackup	Write data into the deep standby backup registers
R_PG_LPC_ReadBackup	Read data from the deep standby backup registers

Register Write Protection Function

Generated Function	Description
R_PG_RWP_RegisterWriteCgc	Enables or disables writing to registers associated with the clock generation circuit
R_PG_RWP_RegisterWriteModeLpcReset	Enables or disables writing to registers associated with the operating mode, low power comsumption, and software reset
R_PG_RWP_RegisterWriteLvd	Enables or disables writing to registers associated with LVD
R_PG_RWP_RegisterWriteMpc	Enables or disables writing to pin-function selection registers
R_PG_RWP_GetStatusCgc	Acquires a value indicating whether writing to registers associated with the clock generation circuit is enabled or disabled
R_PG_RWP_GetStatusModeLpcReset	Acquires a value indicating whether writing to registers associated with the operating mode, low power comsumption, and software reset is enabled or disabled
R_PG_RWP_GetStatusLvd	Acquires a value indicating whether writing to registers associated with LVD is enabled or

	disabled
R_PG_RWP_GetStatusMpc	Acquires a value indicating whether writing to pin-function selection registers is enabled or disabled

Interrupt controller (ICUb)

Generated Function	Description
R_PG_ExtInterrupt_Set_*<interrupt type>*	Set up an external interrupt
R_PG_ExtInterrupt_Disable_*<interrupt type>*	Disable the setting of 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_ExtInterrupt_EnableFilter_*<interrupt type>*	Re-enable the digital filter
R_PG_ExtInterrupt_DisableFilter_*<interrupt type>*	Disable the digital filter
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_PresetBus	Set the bus priority
R_PG_ExtBus_SetBus	Set the bus pins and the bus error monitoring
R_PG_ExtBus_GetErrorStatus	Acquire the status of bus error generation
R_PG_ExtBus_ClearErrorFlags	Clear the bus-error status registers
R_PG_ExtBus_SetArea_CS*<CS area number>*	Set up CS area
R_PG_ExtBus_SetEnable	Enable external bus
R_PG_ExtBus_DisableArea_CS*<CS area number>*	Disable CS area
R_PG_ExtBus_SetDisable	Disable external bus

DMA controller (DMACA)

Generated Function	Description
R_PG_DMAC_Set_C*<channel number>*	Set up a DMAC channel
R_PG_DMAC_Activate_C*<channel number>*	Make the DMAC be ready for the start trigger
R_PG_DMAC_StartTransfer_C*<channel number>*	Start the one transfer of DMAC (Software trigger)
R_PG_DMAC_StartContinuousTransfer_C*<channel number>*	Start the continuous transfer of DMAC (Software trigger)
R_PG_DMAC_StopContinuousTransfer_C*<channel number>*	Stop the software-triggered continuous transfer of DMAC
R_PG_DMAC_Suspend_C*<channel number>*	Suspend the data transfer
R_PG_DMAC_GetTransferCount_C*<channel number>*	Get the transfer counter value
R_PG_DMAC_SetTransferCount_C*<channel number>*	Set the transfer counter
R_PG_DMAC_GetRepeatBlockSizeCount_C*<channel number>*	Get the repeat/block size counter value
R_PG_DMAC_SetRepeatBlockSizeCount_C*<channel number>*	Set the repeat/block size count
R_PG_DMAC_ClearInterruptFlag_C*<channel number>*	Get and clear the interrupt request flag
R_PG_DMAC_GetTransferEndFlag_C*<channel number>*	Get the transfer end flag
R_PG_DMAC_ClearTransferEndFlag_C*<channel number>*	Clear the transfer end flag
R_PG_DMAC_GetTransferEscapeEndFlag_C*<channel number>*	Get the transfer escape end flag

R_PG_DMAC_ClearTransferEscapeEndFlag_C<channel number>	Clear the escape transfer end flag
R_PG_DMAC_SetSrcAddress_C<channel number>	Set the source address
R_PG_DMAC_SetDestAddress_C<channel number>	Set the destination address
R_PG_DMAC_SetAddressOffset_C<channel number>	Set the address offset
R_PG_DMAC_SetExtendedRepeatSrc_C<channel number>	Set the source address extended repeat value
R_PG_DMAC_SetExtendedRepeatDest_C<channel number>	Set the destination address extended repeat value
R_PG_DMAC_StopModule_C<channel number>	Stop the DMAC channel

Data Transfer Controller (DTCa)

Generated Function	Description
R_PG_DTC_Set	Set up the DTC
R_PG_DTC_Set_<trigger source>	Set the DTC transfer data
R_PG_DTC_Activate	Make DTC be ready for the trigger
R_PG_DTC_SuspendTransfer	Stop transfer data
R_PG_DTC_GetTransmitStatus	Get transfer data 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 Port Input Register
R_PG_IO_PORT_Read_P<port number><pin number>	Read 1-bit data from Port Input Register
R_PG_IO_PORT_Write_P<port number>	Write data to Port Output Data Register
R_PG_IO_PORT_Write_P <port number><pin number>	Write 1-bit data to Port Output Data Register
R_PG_IO_PORT_SetPortNotAvailable	Handle unavailable pins

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 of two or more channels 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 MTU counter value
R_PG_Timer_GetRequestFlag_MTU_U<unit number>_C<channel number>	Acquire and clear the MTU interrupt flags
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 value
R_PG_Timer_SetTGR_<general register>_MTU_U<unit number>_C<channel number>	Set the general register value
R_PG_Timer_SetBuffer_AD_MTU_U<unit number>_C<channel	Set A/D converter start request cycle set

number>	buffer registers (TADCOBRA and TADCOBRB)
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 timer output pins in high-impedance state
R_PG_POE_GetRequestFlagHiZ_<*Timer channels/flag*>	Acquire the high-impedance request flags
R_PG_POE_GetShortFlag_<*Timer channels*>	Acquire the MTU output short flags
R_PG_POE_ClearFlag_<*Timer channels/flag*>	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*>	Se 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_<*GTCCRn*>_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_SetCounterValue_GPT_U<*unit number*>_C<*channel number*>	Set the GPT counter value
R_PG_GetCounterValue_GPT_U<*unit number*>_C<*channel number*>	Get the GPT counter value
R_PG_SynchronouslyClearCounter_GPT_U<*unit number*>	Clear the counter of two or more channels simultaneously
R_PG_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*	Set the value to the timer cycle setting buffer

number>_C<channel number>	register (GTPBR)
R_PG_Timer_SetDoubleBuffer_Cycle_GPT_ U<unit number>_C<channel number>	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_<GTDVn>_GPT_ U<unit number>_C<channel number>	Set the value to the timer dead time buffer register U, D (GTDVU, GTDVD)
R_PG_Timer_GetRequestFlag_GPT_ U<unit number>_C<channel number>	Acquire and clear the GPT interrupt flag
R_PG_Timer_GetRequestFlag_GPT_ U<unit number>	Acquire and clear the GPT unit interrupt flag
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

Compare Match Timer (CMT)

Generated Function	Description
R_PG_Timer_Set_CMT_U<unit number>_C<channel number>	Set up the CMT
R_PG_Timer_StartCount_CMT_U<unit number>_C<channel number>	Start or resume the CMT count operation
R_PG_Timer_HaltCount_CMT_U<unit number>_C<channel number>	Halt the CMT count
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_SetConstantRegister_CMT_U<unit number>_C<channel number>	Set the CMT constant register value
R_PG_Timer_StopModule _CMT_U<unit number>	Shut down the CMT unit

Watchdog Timer (WDTA)

Generated Function	Description
R_PG_Timer_Start_WDT	Set up the WDT and start the count
R_PG_Timer_RefreshCounter_WDT	Refresh the counter of WDT
R_PG_Timer_GetStatus_WDT	Acquires the status flag and count value of WDT

Independent Watchdog Timer (IWDTa)

Generated Function	Description
R_PG_Timer_Start_IWDT	Sets up the IWDT and starts its timer
R_PG_Timer_RefreshCounter_IWDT	Refresh the counter
R_PG_Timer_GetStatus_IWDT	Acquires the status flag and count value of IWDT

Serial Communications Interface (SCIc, SCId)

Generated Function	Description
R_PG_SCI_Set_C*<channel number>*	Set a SCI channel
R_PG_SCI_SendTargetStationID_C*<channel number>*	Transmits the ID code of the receiving station
R_PG_SCI_StartSending_C*<channel number>*	Start the data transmission
R_PG_SCI_SendAllData_C*<channel number>*	Transmit all data
R_PG_SCI_I2CMode_Send_C*<channel number>*	Transmit data by simple I^{2}C bus interface
R_PG_SCI_I2CMode_SendWithoutStop_C*<channel number>*	Transmit data by simple I^{2}C bus interface (no stop condition)
R_PG_SCI_I2CMode_GenerateStopCondition_C*<channel number>*	Generate a stop condition
R_PG_SCI_I2CMode_Receive_C*<channel number>*	Receive data by simple I^{2}C bus interface
R_PG_SCI_I2CMode_RestartReceive_C*<channel number>*	Receive data by simple I^{2}C bus interface (RE-START condition)
R_PG_SCI_I2CMode_ReceiveLast_C*<channel number>*	Making reception complete in simple I^{2}C bus interface
R_PG_SCI_I2CMode_GetEvent_C*<channel number>*	Get the detected event in the simple I^{2}C mode
R_PG_SCI_SPIMode_Transfer_C*<channel number>*	Transmit data by simple SPI mode
R_PG_SCI_SPIMode_GetErrorFlag_C*<channel number>*	Get the serial reception error flag in the

	simple SPI mode
R_PG_SCI_GetSentDataCount_C<*channel number*>	Acquire the number of transmitted data
R_PG_SCI_ReceiveStationID_C<*channel number*>	Receives the ID code matches the ID of the receiving station itself
R_PG_SCI_StartReceiving_C<*channel number*>	Start the data reception
R_PG_SCI_ReceiveAllData_C<*channel number*>	Receive all data
R_PG_SCI_ControlClockOutput_C<*channel number*>	Control the output from the SCKn pin (n: 0, 1, 2, 3, or 12)
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_ClearReceptionErrorFlag_C<*channel number*>	Clear the serial reception error flag
R_PG_SCI_GetTransmitStatus_C<*channel number*>	Get the state of transmission
R_PG_SCI_StopModule_C<*channel number*>	Shut down a SCI channel

I^{2}C Bus Interface (RIIC)

Generated Function	Description
R_PG_I2C_Set_C<*channel number*>	Set up the I^{2}C 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 the stop condition
R_PG_I2C_GetBusState_C<*channel number*>	Get the bus state
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 I^{2}C 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_LoopBack<*loopback mode*>_C<*channel number*>	Set loopback mode
R_PG_RSPI_StopModule_C<*channel number*>	Shut down a RSPI channel

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 CRC calculation result
R_PG_CRC_ClearResult	Clear the CRC calculation result
R_PG_CRC_StopModule	Shut down CRC Calculator

12-Bit A/D Converter (S12ADB)

Generated Function	Description
R_PG_ADC_12_Set_S12AD*<unit number>*	Sets up the 12-bit A/D converter
R_PG_ADC_12_StartConversion_S12AD*<unit number>*	Starts A/D conversion
R_PG_ADC_12_StopConversion_S12AD*<unit number>*	Stops A/D conversion
R_PG_ADC_12_GetResult_S12AD*<unit number>*	Gets the result of A/D conversion of an analog input
R_PG_ADC_12_GetResult_SelfDiag_S12AD*<unit number>*	Gets the result of A/D conversion as part of self diagnosis by the A/D Converter
R_PG_ADC_12_StartComparator_S12AD*<unit number>*	Start the comparator
R_PG_ADC_12_StopConparator_S12AD*<unit number>*	Stop the comparator
R_PG_ADC_12_GetComparatorStatusFlag_S12AD *<unit number>*	Get the comparator status flag
R_PG_ADC_12_StopModule_S12AD*<unit number>*	Shuts down the 12-bit A/D converter

10-Bit A/D Converter (AD)

Generated Function	Description
R_PG_ADC_10_Set_AD*<unit number>*	Sets up the 10-bit A/D converter
R_PG_ADC_10_StartConversion_S12AD*<unit number>*	Starts A/D conversion
R_PG_ADC_10_StopConversion_S12AD*<unit number>*	Stops A/D conversion
R_PG_ADC_10_GetResult_S12AD*<unit number>*	Gets the result of A/D conversion of an analog input, output from the temperature sensor, or internal reference voltage
R_PG_ADC_10_GetResult_SelfDiag_S12AD*<unit number>*	Gets the result of A/D conversion as part of self diagnosis by the A/D Converter
R_PG_ADC_10_StopModule_S12AD*<unit number>*	Shuts down the 10-bit A/D converter

D/A Converter (DAa)

Generated Function	Description
R_PG_DAC_Set_C*<channel number>*	Set up a D/A converter channel
R_PG_DAC_SetWithInitialValue_C*<channel number>*	Set up a D/A converter channel and input the data
R_PG_DAC_ControlOutput_C*<channel number>*	Input the data
R_PG_DAC_StopOutput_C*<channel number>*	Stop output

Data Operation Circuit (DOC)

Generated Function	Description
R_PG_DOC_Set	Set up the Data Operation Circuit
R_PG_DOC_GetStatusFlag	Acquire the status of data operation circuit
R_PG_DOC_GetResult	Acquire the result of data operation
R_PG_DOC_InputData	Input data

R_PG_DOC_UpdateData	Update data
R_PG_DOC_StopModule	Shut down Data Operation Circuit

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, R_CGC_Control

Details

- Sets up each clock source and starts the oscillation.
- Switches the internal clock source to the clock which is specified on GUI.
- To insert wait cycles before switching the internal clock source, use R_PG_Clock_WaitSet.

Example

```
//Include "R_PG_<PDG 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_WaitSet

Definition bool R_PG_Clock_WaitSet(void)

Description Set up the clocks (wait cycle insertion)

Parameter

double wait_time	Oscillation stabilization waiting time (in seconds)

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_Clock.c

RPDL function R_CGC_Set, R_CGC_Control

Details

- Sets up each clock source and starts the oscillation.
- Switches the internal clock source to the clock which is specified on GUI.
- This function inserts wait cycles before switching the internal clock source. If wait cycles are not required, use R_PG_Clock_Set.
- The actual waiting time may be different from the specified value.

Example

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

void func(void)
{
	//Set the clock-generation circuit and switch the clock source after waiting 0.5 seconds.
	R_PG_Clock_WaitSet(0.5);
}
```

5.1.3 R_PG_Clock_Start_MAIN

Definition: bool R_PG_Clock_Start_MAIN(void)

Description: Start the main clock oscillator

Conditions for output: The main clock or PLL circuit is set to be used on GUI.

Parameter:

None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Clock.c

RPDL function: R_CGC_Control

Details:

- Starts the main clock oscillator.
- If the main clock is set to be used on GUI, the main clock will start the oscillation in R_PG_Clock_Set.

Example:

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

void func(void)
{
        //Start the main clock oscillator.
        R_PG_Clock_Start_MAIN();
}
```

5.1.4 R_PG_Clock_Stop_MAIN

Definition
bool R_PG_Clock_Stop_MAIN(void)

Description
Stop the main clock oscillator

Conditions for output
The main clock or PLL circuit is set to be used on GUI.

Parameter

None

Return value

true	Setting was made correctly
false	Setting failed

File for output
R_PG_Clock.c

RPDL function
R_CGC_Control

Details

- Stops the main clock oscillator.
- The main clock oscillator cannot be stopped when the main clock or PLL circuit is used as the internal clock source.

Example

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

void func(void)
{
	//Stop the main clock oscillator.
	R_PG_Clock_Stop_MAIN();
}
```

5.1.5 R_PG_Clock_Enable_MAIN_ForcedOscillation

Definition: bool R_PG_Clock_Enable_MAIN_ForcedOscillation(void)

Description: Enable the main clock forced oscillation

Conditions for output: The main clock or PLL circuit is set to be used on GUI.

Parameter

None

Return value

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Clock.c

RPDL function: R_CGC_Control

Details

- Enables the main clock forced oscillation.

Example

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

void func(void)
{
	//Enable the main clock forced oscillation
	R_PG_Clock_Enable_MAIN_ForcedOscillation();
}
```

5.1.6 R_PG_Clock_Disable_MAIN_ForcedOscillation

Definition
bool R_PG_Clock_Disable_MAIN_ForcedOscillation(void)

Description
Disable the main clock forced oscillation

Conditions for output
The main clock or PLL circuit is set to be used on GUI.

Parameter

None

Return value

true	Setting was made correctly
false	Setting failed

File for output
R_PG_Clock.c

RPDL function
R_CGC_Control

Details
- Disables the main clock forced oscillation

Example

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

void func(void)
{
	//Disable the main clock forced oscillation
	R_PG_Clock_Disable_MAIN_ForcedOscillation();
}
```

5.1.7 R_PG_Clock_Start_LOCO

Definition bool R_PG_Clock_Start_LOCO(void)

Description Start the low-speed on-chip oscillator (LOCO)

Parameter

None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_Clock.c

RPDL function R_CGC_Control

Details

- Starts the low-speed on-chip oscillator (LOCO).

Example

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

void func(void)
{
	//Start the low-speed on-chip oscillator (LOCO).
	R_PG_Clock_Start_LOCO();
}
```

5.1.8 R_PG_Clock_Stop_LOCO

Definition bool R_PG_Clock_Stop_LOCO(void)

Description Stop the low-speed on-chip oscillator (LOCO)

Parameter

None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_Clock.c

RPDL function R_CGC_Control

Details

- Stops the low-speed on-chip oscillator (LOCO).
- The low-speed on-chip oscillator (LOCO) cannot be stopped when the LOCO is used as the internal clock source.

Example

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

void func(void)
{
	//Stop the low-speed on-chip oscillator (LOCO).
	R_PG_Clock_Stop_LOCO();
}
```

5.1.9 R_PG_Clock_Start_PLL

Definition bool R_PG_Clock_Start_PLL(void)

Description Start the PLL circuit

Parameter

None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_Clock.c

RPDL function R_CGC_Control

Details

- Starts the PLL circuit.

Example

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

void func(void)
{
	//Start the PLL circuit.
	R_PG_Clock_Start_PLL();
}
```

5.1.10 R_PG_Clock_Stop_PLL

Definition bool R_PG_Clock_Stop_PLL(void)

Description Stop the PLL circuit

Parameter

None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_Clock.c

RPDL function R_CGC_Control

Details

- Stops the PLL circuit.
- The PLL circuit cannot be stopped when the PLL circuit is used as the internal clock source.

Example

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

void func(void)
{
	//Stop the PLL circuit.
	R_PG_Clock_Stop_PLL();
}
```

5.1.11 R_PG_Clock_Enable_BCLK_PinOutput

Definition: bool R_PG_Clock_Enable_BCLK_PinOutput (void)

Description: Enable BCLK pin output

Conditions for output: TheBCLK pin output has been set on GUI.

Parameter:

None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Clock.c

RPDL function: R_CGC_Control

Details:
- Enables clock output from BCLK pin.
- The BCLK clock is output when the external bus is enabled.
- If the BCLK pin output has been set on GUI, the BCLK pin output is enabled in R_PG_Clock_Set.

Example:

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

void func(void)
{
	//Enable BCLK pin output
	R_PG_Clock_Enable_BCLK_PinOutput ();
}
```

5.1.1 R_PG_Clock_Disable_BCLK_PinOutput

Definition: bool R_PG_Clock_Enable_BCLK_PinOutput (void)

Description: Disable BCLK pin output

Conditions for output: The BCLK pin output has been set on GUI.

Parameter:

None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Clock.c

RPDL function: R_CGC_Control

Details:

- Disables clock output from BCLK pin.

Example:

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

void func(void)
{
	//Disable BCLK pin output
	R_PG_Clock_Disable_BCLK_PinOutput ();
}
```

5.1.2 R_PG_Clock_Enable_MAIN_StopDetection

Definition: bool R_PG_Clock_Enable_MAIN_StopDetection(void)

Description: Enable the main clock oscillation stop detection function

Conditions for output: The main clock oscillation stop detection function has been set on GUI.

Parameter:

None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Clock.c

RPDL function: R_CGC_Control

Details:

- Enables the main clock oscillation stop detection function.
- If the main clock oscillation stop detection function has been set on GUI, the function is set up and enabled in R_PG_Clock_Set.

Example:

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

void func(void)
{
	//Enable main clock oscillation stop detection function
	R_PG_Clock_Enable_MAIN_StopDetection();
}
```

5.1.3 R_PG_Clock_Disable_MAIN_StopDetection

Definition: bool R_PG_Clock_Disable_MAIN_StopDetection(void)

Description: Disable the main clock oscillation stop detection function

Conditions for output: The main clock oscillation stop detection function has been set on GUI.

Parameter:

None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Clock.c

RPDL function: R_CGC_Control

Details:

- Disables the main clock oscillation stop detection function.

Example:

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

void func(void)
{
        //Disable main clock oscillation stop detection function
        R_PG_Clock_Disable_MAIN_StopDetection();
}
```

5.1.4 R_PG_Clock_GetFlag_MAIN_StopDetection

Definition: bool R_PG_Clock_GetFlag_MAIN_StopDetection (bool* stop)

Description: Acquire the main clock oscillation stop detection flag

Conditions for output: The main clock oscillation stop detection function has been set on GUI.

Parameter

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

Return value

true	Acquisition of the flag succeeded
false	Acquisition of the flag failed

File for output: R_PG_Clock.c

RPDL function: R_CGC_GetStatus

Details

- Acquires the main clock oscillation stop detection flag.

Example

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

bool stop;

void func(void)
{
	//Acquire the main clock oscillation stop detection flag
	R_PG_Clock_GetFlag_MAIN_StopDetection( &stop );
}
```

5.1.5 R_PG_Clock_ClearFlag_MAIN_StopDetection

Definition: bool R_PG_Clock_ClearFlag_MAIN_StopDetection (void)

Description: Clear the main clock oscillation stop detection flag

Conditions for output: The main clock oscillation stop detection function has been set on GUI.

Parameter:

None

Return value:

true	Clearing succeeded
false	Clearing failed

File for output: R_PG_Clock.c

RPDL function: R_CGC_Control

Details:

- Clears the main clock oscillation stop detection flag.

Example:

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

void func(void)
{
	//Clear the main clock oscillation stop detection flag
	R_PG_Clock_ClearFlag_MAIN_StopDetection();
}
```

5.1.6 R_PG_Clock_GetSelectedClockSource

Definition bool R_PG_Clock_GetSelectedClockSource (uint8_t* clock)

Description Acquire the current internal clock source

Parameter

uint8_t* clock	The address of storage area for the value that corresponds to current internal clock source Correspondence between clock sources and stored values 0:Low-speed on-chip oscillator 2:Main clock 4:PLL circuit

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output R_PG_Clock.c

RPDL function R_CGC_GetStatus

Details
- Acquires the current internal clock source

Example

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

uint8_t clock;

void func(void)
{
	//Acquire the current internal clock source
	R_PG_Clock_GetSelectedClockSource( &clock );
}
```

5.1.7 R_PG_Clock_GetClocksStatus

Definition bool R_PG_Clock_GetClocksStatus (bool* pll, bool* main, bool* loco, bool* iwdt)

Description Acquire the status of the clocks

Parameter

bool* pll	The address of storage area for the value of the PLL stop bit (0:Operating 1:Stopped)
bool* main	The address of storage area for the value of the main clock stop bit (0:Operating 1:Stopped)
bool* loco	The address of storage area for the value of the low-speed on-chip oscillator stop bit (0:Operating 1:Stopped)
bool* iwdt	The address of storage area for the value of the IWDT-dedicated low-speed on-chip oscillator stop bit (0:Operating 1:Stopped)

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output R_PG_Clock.c

RPDL function R_CGC_GetStatus

Details

- Acquire the oscillation status of the clocks
- Specify the address of storage area for the item to be acquired. Specify 0 for a item that is not required.

Example

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

bool loco;

void func(void)
{
    //Acquire the status of the the low-speed on-chip oscillator
    R_PG_Clock_GetClocksStatus ( 0,  0,  &loco,  0 );
}
```

5.2 Voltage Detection Circuit (LVDA)

5.2.1 R_PG_LVD_Set

Definition bool R_PG_LVD_Set (void)

Description Set up the voltage detection circuit (Voltage-monitoring 1 and Voltage-monitoring 2)

Parameter None

Return value

true	Setting was made correctly.
false	Setting failed.

File for output R_PG_LVD.c

RPDL function R_LVD_Create

Details

- This function sets the operation (internal reset or interrupt) when low voltage is detected.
- Both Voltage-monitoring 1 and Voltage-monitoring 2 can be set up in one function call.
- Function R_PG_Clock_Set must be called before any use of this function.

Example

```
//Include "R_PG_<PDG 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.

	// Set up the voltage detection circuit(voltage-monitoring 1 and voltage-monitoring 2)
	R_PG_LVD_Set();
}
```

5.2.2 R_PG_LVD_GetStatus

Definition
bool R_PG_LVD_GetStatus
(bool * lvd1_detect, bool * lvd1_monitor, bool * lvd2_detect, bool * lvd2_monitor)

Description
Get the status flag of Voltage Detection Circuit

Parameter

bool * lvd1_detect	The address of storage area for Voltage Monitoring 1 Voltage Change Detection Flag
bool * lvd1_monitor	The address of storage area for Voltage Monitoring 1 Signal Monitor Flag
bool * lvd2_detect	The address of storage area for Voltage Monitoring 2 Voltage Change Detection Flag
bool * lvd2_monitor	The address of storage area for Voltage Monitoring 2 Signal Monitor Flag

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output
R_PG_LVD.c

RPDL function
R_LVD_GetStatus

Details

- This function acquires the status flag of Voltage Detection Circuit.
- Specify 0 for a flag that is not required.

Example

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

bool lvd1_det, lvd2_det;
bool lvd1_mon, lvd2_mon;

void func(void)
{
	// Get the status flag of Voltage Detection Circuit.
	R_PG_LVD_GetStatus(&lvd1_det, &lvd1_mon, &lvd2_det, &lvd2_mon);

	if( lvd1_det ){
		//Processing when Voltage Monitoring 1 Voltage Change is detected
	}
	if( lvd2_det ){
		//Processing when Voltage Monitoring 2 Voltage Change is detected
	}
}
```

5.2.3 R_PG_LVD_ClearDetectionFlag_LVD<*Voltage Detection Circuit number*>

Definition
bool R_PG_LVD_ClearDetectionFlag_LVD<*Voltage Detection Circuit number*> (void)
<*Voltage Detection Circuit number*>: 1 or 2

Description
Clear Voltage Monitoring n Voltage Change Detection Flag n: 1 or 2

Parameter
None

Return value

true	Clearing succeeded
false	Clearing failed

File for output
R_PG_LVD.c

RPDL function
R_LVD_Control

Details
- This function clears Voltage Monitoring n Voltage Change Detection Flag. n: 1 or 2

Example

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

void func(void)
{
	// Clear Voltage Monitoring 1 Voltage Change Detection Flag.
	R_PG_LVD_ClearDetectionFlag_LVD1();
}
```

5.2.4 R_PG_LVD_Disable_LVD<*Voltage Detection Circuit number*>

Definition: bool R_PG_LVD_Disable_LVD<*Voltage Detection Circuit number*> (void)
<*Voltage Detection Circuit number*>: 1 or 2

Description: Disable Voltage Monitoring n n: 1 or 2

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 disables Voltage Monitoring n. n: 1 or 2

Example:

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

void func(void)
{
	// Disable Voltage Monitoring 1.
	R_PG_LVD_Disable_LVD1();
}
```

5.3 Clock Frequency Accuracy Measurement Circuit (CAC)

5.3.1 R_PG_CAC_Set

Definition bool R_PG_CAC_Set(void)

Description Set up the CAC and start the measurement

Parameter

None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_CAC.c

RPDL function R_CAC_Create

Details
- Sets up the clock frequency accuracy measurement circuit (CAC) and starts the measurement.
- Call R_CGC_Set to set up the clocks before calling this function.

Example

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

 void func(void)
{
	//Set up the clocks
	R_PG_Clock_Set();

	//Set up the CAC and start the measurement
	R_PG_CAC_Set();
}
```

5.3.2 R_PG_CAC_ClearFlag_FrequencyError

Definition

bool R_PG_CAC_ClearFlag_FrequencyError(void)

Description

Clear the frequency error flag

Conditions for output

The frequency error interrupt (FERRF) is set to be enabled on GUI.

Parameter

None

Return value

true	Clearing succeeded
false	Clearing failed

File for output

R_PG_CAC.c

RPDL function

R_CAC_Control

Details

- Clear the frequency error flag.

Example

A case where the setting has been made in the GUI as follows.

- The frequency error interrupt (FERRF) has been set
- CacErrIntFunc has been specified as the frequency error interrupt (FERRF) notification function name

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

void CacErrIntFunc(void)
{
	//Operation when the frequency error interrupt occurs
	func2 ();

	//Clear the frequency error flag
	R_PG_CAC ClearFlag_FrequencyError ();
}

void func1(void)
{
	//Set up the clocks
	R_PG_Clock_Set();

	//Set up the CAC and start the measurement
	R_PG_CAC_Set();
}
```

5.3.3 R_PG_CAC_ClearFlag_MeasurementEnd

Definition bool R_PG_CAC_ClearFlag_MeasurementEnd (void)

Description Clear the measurement end flag

Conditions for output The measurement end interrupt (FENDF) is set to be enabled on GUI.

Parameter

None

Return value

true	Clearing succeeded
false	Clearing failed

File for output R_PG_CAC.c

RPDL function R_CAC_Control

Details
- Clear the measurement end flag.

Example

A case where the setting has been made in the GUI as follows.

- The measurement end interrupt (MENDF) has been set
- CacEndIntFunc has been specified as the measurement end interrupt (MENDF) notification function name

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

void CacEndIntFunc(void)
{
        //Operation when the measurement end interrupt occurs
        func2 ();

        //Clear the measurement end flag
        R_PG_CAC ClearFlag_MeasurementEnd ();
}

void func1(void)
{
        //Set up the clocks
        R_PG_Clock_Set();

        //Set up the CAC and start the measurement
        R_PG_CAC_Set();
}
```

5.3.4 R_PG_CAC_ClearFlag_Overflow

Definition: bool R_PG_CAC_ClearFlag_Overflow (void)

Description: Clear the overflow flag

Conditions for output: The overflow interrupt (OVFF) is set to be enabled on GUI.

Parameter:

None

Return value:

true	Clearing succeeded
false	Clearing failed

File for output: R_PG_CAC.c

RPDL function: R_CAC_Control

Details:

- Clear the overflow flag

Example:

A case where the setting has been made in the GUI as follows.

- The overflow interrupt (OVFF) has been set
- CacOvIntFunc has been specified as the overflow interrupt (OVFF) notification function name

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

void CacOvIntFunc(void)
{
	//Operation when the frequency error interrupt occurs
	func2 ();

	//Clear the overflow flag
	R_PG_CAC ClearFlag_Overflow ();
}

void func1(void)
{
	//Set up the clocks
	R_PG_Clock_Set();

	//Set up the CAC and start the measurement
	R_PG_CAC_Set();
}
```

5.3.5 R_PG_CAC_StartMeasurement

Definition bool R_PG_CAC_StartMeasurement (void)

Description Start the measurement

Parameter

None

Return value

true	Setting was made correntl y
false	Setting failed

File for output R_PG_CAC.c

RPDL function R_CAC_Control

Details
- Resume the measurement which has been stopped by R_PG_CAC_StopMeasurement.

Example

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

void func1(void)
{
        //Stop the measurement
        R_PG_CAC_StopMeasurement ();

        //Clear the frequency error flag
        R_PG_CAC ClearFlag_FrequencyError ();
}

void func2(void)
{
        //Start the measurement
        R_PG_CAC_StartMeasurement();
}
```

5.3.6 R_PG_CAC_StopMeasurement

Definition bool R_PG_CAC_StopMeasurement (void)

Description Stop the measurement

Parameter

None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_CAC.c

RPDL function R_CAC_Control

Details
- Stop the measurement.

Example

Refer to the example of R_PG_CAC_StartMeasurement.

5.3.7 R_PG_CAC_GetStatusFlags

Definition bool R_PG_CAC_GetStatusFlags(bool *err, bool *end, bool *ov)

Description Acquire the CAC status flags

Parameter

bool * err	The address of storage area for the frequency error flag
bool * end	The address of storage area for the measurement end flag
bool * ov	The address of storage area for the overflow flag

Return value

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

File for output R_PG_CAC.c

RPDL function R_CAC_GetStatus

Details

- Acquires the frequency error flag, the measurement end flag and the overflow flag.

Example

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

bool g_err;
bool g_end;
bool g_ov;

void func(void)
{
	//Acquire the CAC status flags
	R_PG_CAC_GetStatusFlags(&g_err, &g_end, &g_ov);
}
```

5.3.8 R_PG_CAC_GetCounterBufferRegister

Definition: bool R_PG_CAC_GetCounterBufferREgicter(uint16_t *cacntbr_val)

Description: Acquire the counter buffer register (CACNTBR) value

Parameter

bool * cacntbr_val	The address of storage area for the counter buffer register (CACNTBR) value

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output: R_PG_CAC.c

RPDL function: R_CAC_GetStatus

Details
- Acquires the couter buffer register (CACNTBR) value.

Example

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

uint16_t cacntbr_val;

void func(void)
{
	//Acquire the CAC status flags
	R_PG_CAC_GetCounterBufferRegister(&cacntbr_val);
}
```

5.3.9 R_PG_CAC_StopModule

Definition bool R_PG_CAC_StopModule(void)

Description Shut down the CAC

Parameter

None

Return value

true	Stopping succeeded
false	Stopping failed

File for output R_PG_CAC.c

RPDL function R_CAC_Destroy

Details

- Shuts down the clock frequency accuracy measurement circuit (CAC).

Example

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

 void func(void)
{
       //Shut down the CAC
       R_PG_CAC_StopModule ();
}
```

5.4 Low Power Consumption

5.4.1 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.
- Call this function before starting the clock source for which you have set the oscillation settling time through the GUI.

Example

```
//Include "R_PG_<PDG 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);
}
```

5.4.2 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_<PDG project name>.h" to use this function.
#include "R_PG_default.h"

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

5.4.3 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_<PDG 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);
}
```

5.4.4 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_<PDG 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);
}
```

5.4.5 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.

Example:

```
//Include "R_PG_<PDG 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 deep software standby mode.
    R_PG_LPC_DeepSoftwareStandby (void);
}
```

5.4.6 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_<PDG 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);
}
```

5.4.7 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 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 reset detection flags and the 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_<PDG 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
	}
}
```

5.4.8 R_PG_LPC_GetLVDDetectionFlag

Definition: bool R_PG_LPC_GetLVDDetectionFlag (bool * lvd0, bool * lvd1, bool * lvd2)

Description: Acquire the value of the LVD detection flags.

Parameter

bool * lvd0	The address of storage area for the LVD0 detection flag
bool * lvd1	The address of storage area for the LVD1 detection flag
bool * lvd2	The address of 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 reset detection flags and the 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_<PDG 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 ( 0, &lvd1, &lvd2 );

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

5.4.9 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 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 reset detection flags and the 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_<PDG 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
	}
}
```

5.4.10 R_PG_LPC_GetStatus

Definition bool R_PG_LPC_GetStatus(uint32_t *data1)

Description Get the status of the low power consumption functions.

Parameter

uint32_t *data1	The address of storage area for the status data 1

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.

data1

<table>
<tr><th>b31-b24</th></tr>
<tr><td>0</td></tr>
</table>

<table>
<tr><th>b23</th><th>b22-b20</th><th>b19</th><th>b18</th><th>b17</th><th>b16</th></tr>
<tr><td colspan="6">Reset status (RSTSR) (0: not detected; 1: detected)</td></tr>
<tr><td>Deep software standby reset</td><td>0</td><td>LVD2</td><td>LVD1</td><td>LVD0</td><td>Power-on reset</td></tr>
</table>

<table>
<tr><th>b15-b13</th><th>b12</th><th>b11-b10</th><th>b9</th><th>b8</th></tr>
<tr><td colspan="5">Deep software standby cancel request detection (DPSIFR) (0: not detected; 1: detected)</td></tr>
<tr><td>0</td><td>NMI</td><td>0</td><td>LVD2</td><td>LVD1</td></tr>
</table>

<table>
<tr><th>b7-b6</th><th>b5</th><th>b4</th><th>b3</th><th>b2</th><th>b1</th><th>b0</th></tr>
<tr><td colspan="7">Deep software standby cancel request detection (DPSIFR) (0: not detected; 1: detected)</td></tr>
<tr><td>0</td><td>IRQ5-DS</td><td>IRQ4-DS</td><td>IRQ3-DS</td><td>IRQ2-DS</td><td>IRQ1-DS</td><td>IRQ0-DS</td></tr>
</table>

- 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_<PDG 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
        }
    }
}
```

5.4.11 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_<PDG 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 );
}
```

5.4.12 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 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_<PDG 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 );
}
```

5.5 Register Write Protection Function

5.5.1 R_PG_RWP_RegisterWriteCgc

Definition bool R_PG_RWP_RegisterWriteCgc (bool enable)

Description Enables or disables writing to registers associated with the clock generation circuit

Parameter

bool enable	Whether writing to registers is enabled or disabled (1: enabled, 0: disabled)

Return value

true	Setting was made correctly.
false	Setting failed.

File for output R_PG_RWP.c

RPDL function R_RWP_Control

Details

- Enables or disables writing to registers associated with the clock generation circuit.

Example

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

bool cgc;
bool mode_lpc_reset;
bool lvd;
bool b0wi,pfswe;

void func1(void)
{
    // Enable writing to registers associated with the clock generation circuit.
    R_PG_RWP_RegisterWriteCgc( 1 );

    // Enable writing to registers associated with the operating mode,
    // low power comsumption, and software reset.
    R_PG_RWP_RegisterWriteModeLpcReset( 1 );

    // Enable writing to registers associated with LVD.
    R_PG_RWP_RegisterWriteLvd( 1 );

    // Enable writing to pin-function selection registers.
    R_PG_RWP_RegisterWriteMpc( 1 );
}

void func2(void)
{
    // Disable writing to registers associated with the clock generation circuit.
    R_PG_RWP_RegisterWriteCgc( 0 );

    // Disable writing to registers associated with the operating mode,
    // low power comsumption, and software reset.
    R_PG_RWP_RegisterWriteModeLpcReset( 0 );

    // Disable writing to registers associated with LVD.
    R_PG_RWP_RegisterWriteLvd( 0 );

    // Disable writing to pin-function selection registers.
    R_PG_RWP_RegisterWriteMpc( 0 );
```

```
}

void func3(void)
{
    // Acquire the value indicating whether writing to registers associated with the clock
    // generation circuit is enabled or disabled.
    R_PG_RWP_GetStatusCgc(&cgc);

    // Acquire the value indicating whether writing to registers associated with
    // the operating mode, low power comsumption, and software reset is enabled or
    // disabled.
    R_PG_RWP_GetStatusModeLpcReset(&mode_lpc_reset);

    // Acquire the value indicating whether writing to registers associated with LVD is
    // enabled or disabled.
    R_PG_RWP_GetStatusLvd(&lvd);

    // Acquire the value indicating whether writing to pin-function selection registers is
    // enabled or disabled.
    R_PG_RWP_GetStatusMpc(&b0wi, &pfswe);
}
```

5.5.2 R_PG_RWP_RegisterWriteModeLpcReset

Definition: bool R_PG_RWP_RegisterWriteModeLpcReset (bool enable)

Description: Enables or disables writing to registers associated with the operating mode, low power comsumption, and software reset

Parameter

bool enable	Whether writing to registers is enabled or disabled (1: enabled, 0: disabled)

Return value

true	Setting was made correctly.
false	Setting failed.

File for output: R_PG_RWP.c

RPDL function: R_RWP_Control

Details:

- Enables or disables writing to registers associated with the operating mode, low power comsumption, and software reset.

Example: Refer to the example of R_PG_RWP_RegisterWriteCgc.

5.5.3 R_PG_RWP_RegisterWriteLvd

Definition bool R_PG_RWP_RegisterWriteLvd (bool enable)

Description Enables or disables writing to registers associated with LVD

Parameter

bool enable	Whether writing to registers is enabled or disabled (1: enabled, 0: disabled)

Return value

true	Setting was made correctly.
false	Setting failed.

File for output R_PG_RWP.c

RPDL function R_RWP_Control

Details
- Enables or disables writing to registers associated with LVD.

Example Refer to the example of R_PG_RWP_RegisterWriteCgc.

5.5.4 R_PG_RWP_RegisterWriteMpc

Definition bool R_PG_RWP_RegisterWriteMpc (bool enable)

Description Enables or disables writing to pin-function selection registers

Parameter

bool enable	Whether writing to registers is enabled or disabled (1: enabled, 0: disabled)

Return value

true	Setting was made correctly.
false	Setting failed.

File for output R_PG_RWP.c

RPDL function R_RWP_Control

Details
- Enables or disables writing to pin-function selection registers.

Example Refer to the example of R_PG_RWP_RegisterWriteCgc.

5.5.5 R_PG_RWP_GetStatusCgc

Definition: bool R_PG_RWP_GetStatusCgc (bool * cgc)

Description: Acquires a value indicating whether writing to registers associated with the clock generation circuit is enabled or disabled

Parameter

bool * cgc	Whether writing to registers associated with the clock generation circuit is enabled or disabled (1: enabled, 0: disabled)

Return value

true	The value of the flag was successfully acquired.
false	Acquisition of the value of the flag failed.

File for output: R_PG_RWP.c

RPDL function: R_RWP_GetStatus

Details:

- Acquires a value indicating whether writing to registers associated with the clock generation circuit is enabled or disabled.

Example: Refer to the example of R_PG_RWP_RegisterWriteCgc.

5.5.6 R_PG_RWP_GetStatusModeLpcReset

Definition bool R_PG_RWP_GetStatusModeLpcReset (bool * mode_lpc_reset)

Description Acquires a value indicating whether writing to registers associated with the operating mode, low power comsumption, and software reset is enabled or disabled

Parameter

bool * mode_lpc_reset	Whether writing to registers associated with the operating mode, low power consumption, and software reset is enabled or disabled (1: enabled, 0: disabled)

Return value

true	The value of the flag was successfully acquired.
false	Acquisition of the value of the flag failed.

File for output R_PG_RWP.c

RPDL function R_RWP_GetStatus

Details
- Acquires a value indicating whether writing to registers associated with the operating mode, low power comsumption, and software reset is enabled or disabled.

Example Refer to the example of R_PG_RWP_RegisterWriteCgc.

5.5.7 R_PG_RWP_GetStatusLvd

Definition bool R_PG_RWP_GetStatusLvd (bool * lvd)

Description Acquires a value indicating whether writing to registers associated with LVD is enabled or disabled

Parameter

bool * lvd	Whether writing to registers associated with LVD is enabled or disabled (1: enabled, 0: disabled)

Return value

true	The value of the flag was successfully acquired.
false	Acquisition of the value of the flag failed.

File for output R_PG_RWP.c

RPDL function R_RWP_GetStatus

Details

- Acquires a value indicating whether writing to registers associated with LVD is enabled or disabled.

Example Refer to the example of R_PG_RWP_RegisterWriteCgc.

5.5.8 R_PG_RWP_GetStatusMpc

Definition bool R_PG_RWP_GetStatusMpc (bool * b0wi, bool * pfswe)

Description Acquires a value indicating whether writing to pin-function selection registers is enabled or disabled

Parameter

bool * b0wi	Whether writing to the PFSWE bit in the PWPR register is enabled or disabled (1: enabled, 0: disabled)
bool * pfswe	Whether writing to the PFS register is enabled or disabled (1: enabled, 0: disabled)

Return value

true	The value of the flag was successfully acquired.
false	Acquisition of the value of the flag failed.

File for output R_PG_RWP.c

RPDL function R_RWP_GetStatus

Details
- Acquires a value indicating whether writing to pin-function selection registers is enabled or disabled.

Example Refer to the example of R_PG_RWP_RegisterWriteCgc.

5.6 Interrupt Controller (ICUb)

5.6.1 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_SetExtInterrupt, R_INTC_CreateExtInterrupt

Details:

- The Multifunction Pin Control registers are modified to enable each selected IRQ pin and the I/O Port PMR and PDR registers are modified to set the pin as an input. For IRQn, the pin to be used 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 this chapter end, 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*>.
- If [Enable digital filter] is specified in the GUI, the digital filter is enabled when called this function.

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

```
//Include "R_PG_<PDG 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_<PDG 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();
}
```

5.6.2 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	Disabling was made correctly
false	Disabling 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 MPC and I/O ports registers for the pin being used for the external interrupt signal are retained.
- When disabling an IRQn pin, the Interrupt Request flag will be cleared automatically.
- When the name of the interrupt notification function has been specified in the GUI, the function having the specified name may be called once more if a valid event occurs just before the interrupt pin is disabled.

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

```
//Include "R_PG_<PDG 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
}
```

5.6.3 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 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: Refer to the Example2 of R_PG_ExtInterrupt_Set_<*interrupt type*>

5.6.4 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 flag succeeded
false	Clearing flag 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 Refer to the Example2 of R_PG_ExtInterrupt_Set_<*interrupt type*>

5.6.5 R_PG_ExtInterrupt_EnableFilter_<*interrupt type*>

Definition
bool R_PG_ExtInterrupt_EnableFilter_<*interrupt type*> (uint32_t div)
<*interrupt type*>: IRQ0 to IRQ7 or NMI

Description
Re-enable the digital filter

Conditions for output
When [Enable digital filter] is specified in the GUI.

Parameter

uint32_t div	Peripheral module clock division values 1: digital filter sampling clock = PCLK 8: digital filter sampling clock = PCLK/8 32: digital filter sampling clock = PCLK/32 64: digital filter sampling clock = PCLK/64

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_ControlExtInterrupt

Details
- The digital filter disabled by R_PG_ExtInterrupt_DisableFilter_<*interrupt type*> is enabled, and digital filter sampling clock is set again.

Example
When [Use IRQ0] is specified in the GUI ([Enable digital filter] is specified)

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

void func1(void)
{
	R_PG_Clock_Set();    //The clock-generation circuit has to be set first.
	R_PG_ExtInterrupt_Set_IRQ0();    //Set IRQ0 (Enabling digital filter)
}

void func2(void)
{
	R_PG_ExtInterrupt_DisableFilter_IRQ0();    //Disabling digital filter

	R_PG_ExtInterrupt_EnableFilter_IRQ0( 1 );    //Re-enabling the digital filter
}
```

5.6.6 R_PG_ExtInterrupt_DisableFilter_<*interrupt type*>

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

Description: Disable the digital filter

Conditions for output: When [Enable digital filter] is specified in the GUI.

Parameter: None

Return value:

true	Disabling was made correctly
false	Disabling failed

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

RPDL function: R_INTC_ControlExtInterrupt

Details:
- The digital filter is disabled.
- Disable the digital filter before transition to Software Standby Mode. To use the digital filter again after return from software standby mode, call R_PG_ExtInterrupt_EnableFilter_<*interrupt type*>.

Example: Refer to the example of R_PG_ExtInterrupt_EnableFilter_<*interrupt type*>

5.6.7 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_<PDG 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
}
```

5.6.8 R_PG_SoftwareInterrupt_Generate

Definition bool R_PG_SoftwareInterrupt_Generate(void)

Description Generate the software interrupt

Parameter None

Return value

true	Generating was made correctly
false	Generating 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 Refer to the example of R_PG_SoftwareInterrupt_Set

5.6.9 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_<PDG 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();
}
```

5.6.10 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 this chapter end, 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

```
//Include "R_PG_<PDG 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
}
```

5.7 Buses

5.7.1 R_PG_ExtBus_PresetBus

Definition bool R_PG_ExtBus_PresetBus(void)

Description Set the bus priority

Conditions for output The bus priority has been set on GUI

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_ExtBus.c

RPDL function R_BSC_Set

Details
- Sets the bus priority.
- If required, call this function before calling R_PG_ExtBus_SetBus.

Example

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

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

5.7.2 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 the bus error interrupt has been set to be enabled on 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 the section Notes on Notification Functions.
- The status of bus error generation can be acquired by calling R_PG_ExtBus_GetErrorStatus.
- The external bus clock (BCLK) can be set by R_PG_Clock_Set.
- If required, call R_PG_ExtBus_PresetBus before calling this function.

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

```
//Include "R_PG_<PDG 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, 0, &master, &err_addr);
	if( addr_err ){
		//Processing when illegal address access error occurs
	}

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

5.7.3 R_PG_ExtBus_GetErrorStatus

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

Description
Acquire the status of bus error generation

Conditions for output
The bus error monitoring has been set on GUI

Parameter

bool * addr_err	The address of storage area for the illegal address access error flag
bool * time_err	The address of storage area for the timeout error flag
uint8_t * master	The address of 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:DMAC/DTC
uint16_t * err_addr	The address of 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
A case where BusErrFunc has been specified as the name of the bus error interrupt notification function.

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

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

//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, 0, &master, &err_addr);
	if( addr_err ){
		//Processing when illegal address access error occurs
	}

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

5.7.4 R_PG_ExtBus_ClearErrorFlags

Definition bool R_PG_ExtBus_ClearErrorFlags(void)

Description Clear the bus-error status registers

Conditions for output The bus error monitoring has been set on GUI

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, timeout error flag, ID code of bus master and a value of accessed address).
- The interrupt request flag (IR flag) is cleared in this function.

Example Refer to the example of R_PG_ExtBus_GetErrorStatus

5.7.5 R_PG_ExtBus_SetArea_CS<*CS area number*>

Definition　bool R_PG_ExtBus_SetArea_CS<*CS area number*>(void)

<*CS area number*> : 0 to 3

Description　Set up CS area

Conditions for output　External area has been set on GUI

Parameter　None

Return value

true	Setting was made correctly
false	Setting failed

File for output　R_PG_ExtBus_CS<*CS area number*>.c

<*CS area number*> : 0 to 3

RPDL function　R_BSC_CreateArea

Details

- Sets up CS area.
- Call R_PG_ExtBus_SetBus before calling this function to set up the bus pins and the bus error monitoring.

Example　A case where CS1 and CS2 are set up.

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

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

	//Set up CS1
	R_PG_ExtBus_SetArea_CS1();

	//Set up CS2
	R_PG_ExtBus_SetArea_CS2();

	//Enable the external bus
	R_PG_ExtBus_SetEnable ();
}
```

5.7.6 R_PG_ExtBus_SetEnable

Definition bool R_PG_ExtBus_SetEnable (void)

Description Enable external bus

Conditions for output External area has been set on GUI

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_ExtBus.c

RPDL function R_BSC_Control

Details

- Enable the external bus.
- Call R_PG_ExtBus_SetBus and R_PG_ExtBus_SetArea_CS<CS area number> to set up the bus pins, the bus error monitoring and CS area before calling this function.

Example Refer to the example of R_PG_ExtBus_SetArea_CS<*CS area number*>.

5.7.7 R_PG_ExtBus_DisableArea_CS<*CS area number*>

Definition bool R_PG_ExtBus_DisableArea_CS<*CS area number*>(void)

<*CS area number*> : 0 to 3

Description Set up CS area

Conditions for output External area has been set on GUI

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_ExtBus_CS<*CS area number*>.c

<*CS area number*> : 0 to 3

RPDL function R_BSC_Destroy

Details

- Disable CS area.

Example

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

void func1(void)
{
	//Set up the bus pins and bus error monitoring.
	R_PG_ExtBus_SetBus();

	//Set up CS0
	R_PG_ExtBus_SetArea_CS0();
}

void func2(void)
{
	//Disable CS0
	R_PG_ExtBus_DisableArea_CS0();
}
```

5.7.8 R_PG_ExtBus_SetDisable

Definition bool R_PG_ExtBus_SetDisable

Description Disable the external bus

Conditions for output External area has been set on GUI

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_ExtBus.c

RPDL function R_BSC_Control

Details

- Disable the external bus.

Example

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

void func(void)
{
	//Disable the external bus
	R_PG_ExtBus_SetDisable ();
}
```

5.8 DMA controller (DMACA)

5.8.1 R_PG_DMAC_Set_C<*channel number*>

Definition: bool R_PG_DMAC_Set_C<*channel number*> (void)
<*channel number*>: 0 to 3

Description: Set up a DMAC channel

Parameter: None

Return value:

true	Setting was made correctly.
false	Setting failed.

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

RPDL function: R_DMAC_Create

Details:

- Releases the DMAC from the module-stop and makes initial settings.
- If an interrupt was selected as a transfer start trigger, the DMAC channel will be ready for the interrupt signal by calling R_PG_DMAC_Activate_C<*channel number*> after calling this function. If the software trigger was selected as a transfer start trigger, DMAC channel will start the data transfer when calling R_PG_DMAC_StartTransfer_C<*channel number*> or R_PG_DMAC_StartContinuousTransfer_C<*channel number*> after calling this function.
- The DMAC interrupt is set by this function. When the name of the interrupt notification function has been specified in the GUI, if a CPU interrupt occurs, 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 this chapter end, Notes on Notification Functions.
- To transfer the SCI transmission data by DMAC, make the following settings.

  DMAC settings

Transfer request source	: TXI0 (SCI0 transmit data empty interrupt)
Operation when the transfer completes	: Clear the interrupt flag of the activation source
Destination start address *Destination start address can be set also from the program. Refer the usage example 2 and 3.	: Address of Transmit Data Register (TDR)
Destination address update mode	: Fixed
Length of a single data	: 1 byte

  SCIc setting

Data transmission method	: Transfer the transmitted serial data by DMAC

  For usage of function, refer to example 2.

- To transfer the SCI reception data by DMAC, make the following settings.

  DMAC settings

Transfer request source	: RXI0 (SCI0 receive data full interrupt)
Operation when the transfer completes	: Clear the interrupt flag of the activation source
Source start address *Source start address can be set also from the program. Refer the usage example 2 and 3.	: Address of Receive Data Register (RDR)
Source address update mode	: Fixed
Length of a single data	: 1 byte

SCIc setting

Data transmission method	: Transfer the received serial data by DMAC

For usage of function, refer to example 3.

Example 1 A case where IRQ0 activates DMA transfer

- IRQ0 interrupt was selected as a transfer start trigger of DMAC0 in GUI.
- Dmac0IntFunc was specified as the DMA interrupt notification function name in the GUI.
- DMAC was selected as an interrupt request destination for IRQ0.

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

void func(void)
{
	R_PG_DMAC_Set_C0();	//Set up DMAC0

	R_PG_ExtInterrupt_Set_IRQ0();	//Set up IRQ0

	R_PG_DMAC_Activate_C0();	//Make DMAC0 be ready for the transfer start trigger
}

//DMA interrupt notification function
void Dmac0IntFunc (void)
{
	R_PG_DMAC_StopModule_C0();	//Stop DMAC
}
```

Example 2 A case where the SCI transmission data is transferred by DMAC

- Dmac0IntFunc was specified as the DMA interrupt notification function name in the GUI.
- The SCI0 transmit data empty interrupt is selected as a DMA transfer trigger.

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

volatile bool sci_dma_transfer_complete;	//DMA transfer end flag
uint8_t tr[]=”ABCDEFG”;	//Data source

void func(void)
{
	//Initialize DMA transfer end flag
	sci_dma_transfer_complete = false;

	R_PG_Clock_Set();	//The clock-generation circuit has to be set first.

	R_PG_SCI_Set_C0();	//Set up SCI0

	R_PG_DMAC_Set_C0();	//Set up DMAC0

	//Set source address, destination address and transfer counter
	R_PG_DMAC_SetSrcAddress_C0( tr );
	R_PG_DMAC_SetDestAddress_C0((void*)&(SCI0.TDR));
	R_PG_DMAC_SetTransferCount_C0( 8 );

	//Make DMAC0 be ready for the transfer start trigger
	R_PG_DMAC_Activate_C0();

	//Enable the SCI0 transmission (TXI interrupt occurs and DMA transfer starts)
	R_PG_SCI_SendAllData_C0(
		PDL_NO_PTR,
		PDL_NO_DATA
	);
	// Wait for the DMAC to complete the transfer
	while (sci_dma_transfer_complete == false);
}

//DMA interrupt notification function
```

```
void Dmac0IntFunc (void)
{
    //SCI transmit end flag
    bool sci_transfer_complete;
    sci_transfer_complete = false;

   // Wait for the SCI to complete the transmission
    do{
        R_PG_SCI_GetTransmitStatus_C0( &sci_transfer_complete );
    } while( ! sci_transfer_complete );

    //Stop the SCI
    R_PG_SCI_StopCommunication_C0();

    //Stop the DMAC
    R_PG_DMAC_StopModule_C0();

    sci_dma_transfer_complete = true;
}
```

Example 3 A case where the SCI reception data is transferred by DMAC

- Dmac0IntFunc was specified as the DMA interrupt notification function name in the GUI.
- The SCI0 receive data empty interrupt is selelcted as a DMA transfer trigger.

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

volatile bool sci_dma_transfer_complete;    //DMA transfer end flag
uint8_t re[]="--------";    //Data destination

void func(void)
{
    //Initialize DMA transfer end flag
    sci_dma_transfer_complete = false;

    R_PG_SCI_Set_C0();    //Set up SCI0

    R_PG_DMAC_Set_C0();    //Set up DMAC0

   //Set source address, destination address and transfer counter
   R_PG_DMAC_SetSrcAddress_C0((void*)&(SCI0.RDR) );
   R_PG_DMAC_SetDestAddress_C0( re );
   R_PG_DMAC_SetTransferCount_C0( 8 );

    //Make DMAC0 be ready for the transfer start trigger
    R_PG_DMAC_Activate_C0();

    //Enable the SCI0 reception
    R_PG_SCI_ReceiveAllData_C0(
        PDL_NO_PTR,
        PDL_NO_DATA
    );
}

//DMA interrupt notification function
void Dmac0IntFunc (void)
{
    //Stop the SCI reception
    R_PG_SCI_StopCommunication_C0();

   //Stop the DMAC
    R_PG_DMAC_StopModule_C0();
}
```

5.8.2 R_PG_DMAC_Activate_C<*channel number*>

Definition
bool R_PG_DMAC_Activate_C<*channel number*> (void)
< *channel number* > : 0 to 3

Description
Make the DMAC be ready for the start trigger

Conditions for output
An interrupt is selected as a transfer start trigger

Parameter
None

Return value

true	Setting was made correctly.
false	Setting failed.

File for output
R_PG_DMAC_C <*channel number*>.c
<*channel number*>: 0 to 3

RPDL function
R_DMAC_Control

Details

- This function makes the DMAC channel be ready for the transfer start trigger.
- This function is genetarted when an interrupt is selected as a transfer start trigger.
- Call R_PG_DMAC_Set_C<*channel number*> to set up a DMAC channel before calling this function.

Example
A case where the setting is made as follows.

- IRQ0 was selected as a transfer start trigger of DMAC0 in normal transfer mode
- Dmac0IntFunc was specified as the DMA0 interrupt notification function name

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

void func(void)
{
    //Set up DMAC0
    R_PG_DMAC_Set_C0();

    //Set IRQ0
    R_PG_ExtInterrupt_Set_IRQ0();

    //Make DMAC0 be ready for the transfer start trigger
    R_PG_DMAC_Activate_C0();
}

//DMA interrupt notification function
void Dmac0IntFunc (void)
{
    //Stop the DMAC
    R_PG_DMAC_StopModule_C0();
}
```

5.8.3 R_PG_DMAC_StartTransfer_C<*channel number*>

Definition
: bool R_PG_DMAC_StartTransfer_C<*channel number*> (void)
< *channel number* > : 0 to 3

Description
: Start the one transfer of DMAC (Software trigger)

Conditions for output
: The software trigger is selected as a transfer start trigger

Parameter
: None

Return value

true	Setting was made correctly.
false	Setting failed.

File for output
: R_PG_DMAC_C <*channel number*>.c
<*channel number*>: 0 to 3

RPDL function
: R_DMAC_Control

Details

- This function starts DMA transfer of the channel specified the software trigger as a transfer start trigger.
- A DMA transfer request is cleared automatically when data transfer is started.

Example

A case where the setting is made as follows.

- The software trigger was selected as a transfer start trigger of DMAC0 in normal transfer mode
- Dmac0IntFunc was specified as the DMA interrupt notification function name

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

volatile bool transferred;

void func(void)
{
	transferred = false;

	//Set up DMAC0
	R_PG_DMAC_Set_C0();

	while( transferred == false ){
		//Start the DMA transfer of DMAC0
		R_PG_DMAC_StartTransfer_C0();
	}
	//Stop the DMAC
	R_PG_DMAC_StopModule_C0();
}

//DMA interrupt notification function
void Dmac0IntFunc (void)
{
	transferred = true;
}
```

5.8.4 R_PG_DMAC_StartContinuousTransfer_C<*channel number*>

Definition: bool R_PG_DMAC_StartContinuousTransfer_C<*channel number*> (void)
< *channel number* > : 0 to 3

Description: Start the continuous transfer of DMAC (Software trigger)

Conditions for output: The software trigger is selected as a transfer start trigger

Parameter: None

Return value:

true	Setting was made correctly.
false	Setting failed.

File for output: R_PG_DMAC_C <*channel number*>.c
<*channel number*>: 0 to 3

RPDL function: R_DMAC_Control

Details:
- This function starts DMA transfer of the channel specified the software trigger as a transfer start trigger.
- This function enables continuous DMA transfer because a DMA transfer request is generated again after completion of a transfer.

Example: A case where the setting is made as follows.
- The software trigger was selected as a transfer start trigger of DMAC0 in normal transfer mode
- Dmac0IntFunc was specified as the DMA interrupt notification function name

```
#include "R_PG_default.h" //Include "R_PG_<PDG project name>.h" to use this function.
void func(void)
{
    //Set up DMAC0
    R_PG_DMAC_Set_C0();

    //Start the DMA transfer of DMAC0
    R_PG_DMAC_StartContinuousTransfer_C0();
}

//DMA interrupt notification function
void Dmac0IntFunc (void)
{
    //Stop the DMAC
    R_PG_DMAC_StopModule_C0();
}
```

5.8.5 R_PG_DMAC_StopContinuousTransfer_C<*channel number*>

Definition: bool R_PG_DMAC_StopContinuousTransfer_C<*channel number*> (void)
< *channel number* > : 0 to 3

Description: Stop the software-triggered continuous transfer of DMAC

Conditions for output: The software trigger is selected as a transfer start trigger

Parameter: None

Return value:

true	Setting was made correctly.
false	Setting failed.

File for output: R_PG_DMAC_C <*channel number*>.c
<*channel number*>: 0 to 3

RPDL function: R_DMAC_Control

Details:
- This function clears DMA transfer request of the channel specified the software trigger as a transfer start trigger.

Example: A case where the setting is made as follows.
- The software trigger was selected as a transfer start trigger of DMAC0 in normal transfer mode

```
#include "R_PG_default.h" //Include "R_PG_<PDG project name>.h" to use this function.
void func1(void)
{
	//Set up DMAC0
	R_PG_DMAC_Set_C0();
	//Start the DMA transfer of DMAC0
	R_PG_DMAC_StartContinuousTransfer_C0();
}
void func2(void)
{
	//Clear DMA transfer request by software
	R_PG_DMAC_StopContinuousTransfer_C0();
}
```

5.8.6 R_PG_DMAC_Suspend_C<*channel number*>

Definition
bool R_PG_DMAC_Suspend_C<*channel number*> (void)
< *channel number* > : 0 to 3

Description
Suspend the data transfer

Parameter
None

Return value

true	Suspending succeeded.
false	Suspending failed.

File for output
R_PG_DMAC_C <*channel number*>.c
<*channel number*>: 0 to 3

RPDL function
R_DMAC_Control

Details
- This function suspends(disables) the DMA transfer.
- This function can suspend the DMA transfer triggered by hardware.
- To resume the transfer, when interrupt is selected as a transfer start trigger, clear the interrupt request flag of trigger source and call R_PG_DMAC_Activate_C<*channel number*> to make the DMAC channel be ready for the transfer start trigger.

Example
A case where the setting is made as follows.
- IRQ0 interrupt was selected as a transfer start trigger of DMAC0 in normal transfer mode
- Dmac0IntFunc was specified as the DMA interrupt notification function name
- Irq1IntFunc was specified as the IRQ1 interrupt notification function name
- Irq2IntFunc was specified as the IRQ2 interrupt notification function name

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

void func(void)
{
    R_PG_DMAC_Set_C0();     //Set up DMAC0
    R_PG_ExtInterrupt_Set_IRQ0();     //Set IRQ0
    R_PG_ExtInterrupt_Set_IRQ1();     //Set IRQ1
    R_PG_ExtInterrupt_Set_IRQ2();     //Set IRQ2
    R_PG_DMAC_Activate_C0();     // Make DMAC0 be ready for the transfer start trigger
}

//DMA interrupt notification function
void Dmac0IntFunc (void)
{
    R_PG_DMAC_StopModule_C0();     //Stop the DMAC
}

//DMA transfer is suspended by IRQ1 input
void Irq1IntFunc (void)
{
    R_PG_DMAC_Suspend_C0();     //Suspend the DMA transfer
}

//DMA transfer is re-activated by IRQ2 input
void Irq2IntFunc (void)
{
    R_PG_ExtInterrupt_ClearRequestFlag_IRQ0();     //Clear the request flag of trigger
    R_PG_DMAC_Activate_C0();     // Make DMAC0 be ready for the transfer start trigger
}
```

5.8.7 R_PG_DMAC_GetTransferCount_C<*channel number*>

Definition
bool R_PG_DMAC_GetTransferCount_C<*channel number*> (uint16_t * count)
< *channel number* > : 0 to 3

Description
Get the transfer counter value

Parameter

uint16_t * count	The address of storage area for the counter value

Return value

true	Acquisition succeeded
false	Acquisition failed.

File for output
R_PG_DMAC_C <*channel number*>.c
<*channel number*>: 0 to 3

RPDL function
R_DMAC_GetStatus

Details
- This function gets the current transfer counter value.
- The DMA interrupt request flag (IR flag) is cleared in this function. Call R_PG_DMAC_ClearInterruptFlag_C<*channel number*> to get the DMA interrupt request flag before calling this function if needed.

Example
A case where the setting is made as follows.
- The transfer start trigger of DMAC0 is interrupt

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

void func(void)
{
      uint16_t count;

      //Set up DMAC0
      R_PG_DMAC_Set_C0();

      //Make DMAC0 be ready for the transfer start trigger
      R_PG_DMAC_Activate_C0();

      //Wait for the transfer counter to become lower than 10
      do{
            R_PG_DMAC_GetTransferCount_C0( & count );
      } while( count >= 10 );

      //Suspend the DMA transfer
      R_PG_DMAC_Suspend_C0();
}
```

5.8.8 R_PG_DMAC_SetTransferCount_C<*channel number*>

Definition: bool R_PG_DMAC_SetTransferCount_C<*channel number*>(uint16_t count)
< *channel number* > : 0 to 3

Description: Set the transfer counter

Parameter:

uint16_t count	Value to be set to the transfer counter

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_DMAC_C<*channel number*>.c
<*channel number*>: 0 to 3

RPDL function: R_DMAC_Control

Details:

- This function sets the transfer counter.
- The valid range of the counter value is from 0 to 65535 (0 : free running mode) in normal
- transfer mode, 0 to 1023 (0 = 1024 units) in repeat transfer mode and block transfer mode.

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

- IRQ0 interrupt was selected as a transfer start trigger of DMAC0
- Dmac0IntFunc was specified as the DMA interrupt notification function name

```
#include "R_PG_default.h" //Include "R_PG_<PDG project name>.h" to use this function.
void func(void)
{
        //Set up DMAC0
        R_PG_DMAC_Set_C0();

        //Set IRQ0
        R_PG_ExtInterrupt_Set_IRQ0();

        // Make DMAC0 be ready for the transfer start trigger
        R_PG_DMAC_Activate_C0();
}
//DMA interrupt notification function
void Dmac0IntFunc (void)
{
       //Suspend the DMA transfer
       R_PG_DMAC_Suspend_C0();

        //Change the DMAC0 settings
        R_PG_DMAC_SetSrcAddress_C0( src_address );    //Source address
        R_PG_DMAC_SetDestAddress_C0( dest_address );    //Destination address
        R_PG_DMAC_SetTransferCount_C0( tr_count );    //Transfer counter

        // Make DMAC0 be ready for the transfer start trigger
        R_PG_DMAC_Activate_C0();
}
```

5.8.9 R_PG_DMAC_GetRepeatBlockSizeCount_C<*channel number*>

Definition
: bool R_PG_DMAC_GetRepeatBlockSizeCount_C<*channel number*> (uint16_t * count)
< *channel number* > : 0 to 3

Description
: Get the repeat/block size counter value

Conditions for output
: Repeat transfer mode or block transfer mode is selected for the transfer mode.

Parameter

uint16_t * count	The address of storage area for the counter value

Return value

true	Acquisition succeeded
false	Acquisition failed.

File for output
: R_PG_DMAC_C <*channel number*>.c
<*channel number*>: 0 to 3

RPDL function
: R_DMAC_GetStatus

Details

- This function gets the current repeat/block size counter value.
- The DMA interrupt request flag (IR flag) is cleared in this function. Call R_PG_DMAC_ClearInterruptFlag_C<*channel number*> to get the DMA interrupt request flag before calling this function if needed.

Example

A case where the setting is made as follows.

- DMAC0 is set to repeat transfer mode
- The transfer start trigger is interrupt

```
#include "R_PG_default.h" //Include "R_PG_<PDG project name>.h" to use this function.
void func(void)
{
    uint16_t count;

    //Set up DMAC0
    R_PG_DMAC_Set_C0();

    //Make DMAC0 be ready for the transfer start trigger
    R_PG_DMAC_Activate_C0();

    //Wait for the repeat size counter to become lower than 10
    do{
        R_PG_DMAC_GetRepeatBlockSizeCount_C0( & count );
    } while( count >= 10 );

    //Suspend the DMA transfer
    R_PG_DMAC_Suspend_C0();
}
```

5.8.10 R_PG_DMAC_SetRepeatBlockSizeCount_C<*channel number*>

Definition
bool R_PG_DMAC_SetRepeatBlockSizeCount_C<*channel number*> (uint16_t count)
< *channel number* > : 0 to 3

Description
Set the repeat/block size counter value

Conditions for output
Repeat transfer mode or block transfer mode is selected for the transfer mode.

Parameter

uint16_t count	Value to be set to the repeat/block size counter

Return value

true	Setting was made correctly
false	Setting failed

File for output
R_PG_DMAC_C <*channel number*>.c
<*channel number*>: 0 to 3

RPDL function
R_DMAC_GetStatus

Details

- This function sets the repeat/block size counter.
  The valid range of the counter value is from 0 to 1023 (0 = 1024 units) in repeat transfer mode, 1 to 1023 in block transfer mode.

Example
A case where the setting is made as follows.

- DMAC0 is set to repeat transfer mode
- IRQ0 interrupt was selected as a transfer start trigger
- Dmac0IntFunc was specified as the DMA interrupt notification function name

```
#include "R_PG_default.h" //Include "R_PG_<PDG project name>.h" to use this function.
void func(void)
{
    //Set up DMAC0
    R_PG_DMAC_Set_C0();

    //Set IRQ0
    R_PG_ExtInterrupt_Set_IRQ0();

    // Make DMAC0 be ready for the transfer start trigger
    R_PG_DMAC_Activate_C0();
}
//DMA interrupt notification function
void Dmac0IntFunc (void)
{
    //Suspend the DMA transfer
    R_PG_DMAC_Suspend_C0();

    //Change the DMAC0 settings
    R_PG_DMAC_SetTransferCount_C0( tr_count );    //Transfer counter
    R_PG_DMAC_SetRepeatBlockSizeCount_C0( repeat_count );    //Repeat size counter

    // Make DMAC0 be ready for the transfer start trigger
    R_PG_DMAC_Activate_C0();
}
```

5.8.11 R_PG_DMAC_ClearInterruptFlag_C<*channel number*>

Definition: bool R_PG_DMAC_ClearInterruptFlag_C<*channel number*> (bool * int_request)
< *channel number* > : 0 to 3

Description: Get and clear the interrupt request flag

Conditions for output: DMA interrupt is enabled

Parameter:

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

Return value:

true	Acquisition and clearing succeeded
false	Acquisition and clearing failed

File for output: R_PG_DMAC_C <*channel number*>.c
<*channel number*>: 0 to 3

RPDL function: R_DMAC_GetStatus

Details:

- This function gets and clears the DMA interrupt request flag (IR flag).

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

- DMAC0 is set to mormal transfer mode
- The transfer start trigger is interrupt
- The DMA interrupt is enabled
- The DMA interrupt priority level is 0

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

void func(void)
{
    bool int_request;

    //Set up DMAC0
    R_PG_DMAC_Set_C0();

    //Make DMAC0 be ready for the transfer start trigger
    R_PG_DMAC_Activate_C0();

    //Wait for the IR flag to become 1
    do{
        R_PG_DMAC_ClearInterruptFlag_C0(& int_request );
    } while( int_request == false );
}
```

5.8.12 R_PG_DMAC_GetTransferEndFlag_C<*channel number*>

Definition
bool R_PG_DMAC_GetTransferEndFlag_C<*channel number*> (bool* end)
< *channel number* > : 0 to 3

Description
Get the transfer end flag

Parameter

bool* end	The address of storage area for the transfer end flag

Return value

true	Acquisition succeeded
false	Acquisition failed.

File for output
R_PG_DMAC_C <*channel number*>.c
<*channel number*>: 0 to 3

RPDL function
R_DMAC_GetStatus

Details

- This function gets the transfer end flag.
- The DMA interrupt request flag (IR flag) is cleared in this function. Call R_PG_DMAC_ClearInterruptFlag_C<*channel number*> to get the DMA interrupt request flag before calling this function if needed.
- The transfer end flag is not cleared in this function. Call R_PG_DMAC_ClearTransferEndFlag_C<*channel number*> to clear the transfer end flag if needed.

Example
A case where the setting is made as follows.

- DMAC0 is set to normal transfer mode
- The transfer start trigger is interrupt
- The DMA interrupt is not enabled

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

void func(void)
{
    bool end;

    //Set up DMAC0
    R_PG_DMAC_Set_C0();

    //Make DMAC0 be ready for the transfer start trigger
    R_PG_DMAC_Activate_C0();

    //Wait for the transfer end flag to become 1
    do{
        R_PG_DMAC_GetTransferEndFlag_C0( & end );
    } while( end == false );

    //Clear the DMA transfer end flag
    R_PG_DMAC_ClearTransferEndFlag_C0();
}
```

5.8.13 R_PG_DMAC_ClearTransferEndFlag_C<*channel number*>

Definition
: bool R_PG_DMAC_ClearTransferEndFlag_C<*channel number*> (void)
  < *channel number* > : 0 to 3

Description
: Clear the transfer end flag

Parameter
: None

Return value

true	Clearing succeeded
false	Clearing failed

File for output
: R_PG_DMAC_C <*channel number*>.c
  <*channel number*>: 0 to 3

RPDL function
: R_DMAC_Control

Details

- This function clears the transfer end flag.
- To get the transfer end flag, call R_PG_DMAC_GetTransferEndFlag_C<*channel number*>.

Example

A case where the setting is made as follows.

- DMAC0 is set to mormal transfer mode
- The transfer start trigger is interrupt
- The DMA interrupt is not enabled

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

void func(void)
{
    bool end;

    //Set up DMAC0
    R_PG_DMAC_Set_C0();

    //Make DMAC0 be ready for the transfer start trigger
    R_PG_DMAC_Activate_C0();

    //Wait for the transfer end flag to become 1
    do{
        R_PG_DMAC_GetTransferEndFlag_C0( & end );
    } while( end == false );

    //Clear the DMA transfer end flag
    R_PG_DMAC_ClearTransferEndFlag_C0();
}
```

5.8.14 R_PG_DMAC_GetTransferEscapeEndFlag_C<*channel number*>

Definition
bool R_PG_DMAC_GetTransferEscapeEndFlag_C<*channel number*> (bool* end)
< *channel number* > : 0 to 3

Description
Get the transfer escape end flag

Conditions for output
[Completion of a 1-block/repeat size transfer], [Source address extended repeat area overflow] or [Destination address extended repeat area overflow] is selected as the interrupt output source

Parameter

bool* end	The address of storage area for the transfer escape end flag

Return value

true	Acquisition succeeded
false	Acquisition failed.

File for output
R_PG_DMAC_C <*channel number*>.c
<*channel number*>: 0 to 3

RPDL function
R_DMAC_GetStatus

Details
- This function gets the DMA transfer escape end flag (EDMSTS.ESIF).
- The DMA interrupt request flag (IR flag) is cleared in this function. Call R_PG_DMAC_ClearInterruptFlag_C<*channel number*> to get the DMA interrupt request flag before calling this function if needed.
- The transfer escape end flag is not cleared in this function. Call R_PG_DMAC_ClearTransferEscapeEndFlag_C<*channel number*> to clear the transfer escape end flag if needed.

Example
A case where the setting is made as follows.
- DMAC0 is set to repeat transfer mode
- The transfer start trigger is interrupt
- [Completion of a 1-block/repeat size transfer] is selected for the interrupt output source
- The DMA interrupt priority level is 0

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

void func(void)
{
    bool end;

    //Set up DMAC0
    R_PG_DMAC_Set_C0();

    //Make DMAC0 be ready for the transfer start trigger
    R_PG_DMAC_Activate_C0();

    //Wait for the transfer escape end flag to become 1
    do{
        R_PG_DMAC_GetTransferEscapeEndFlag_C0( & end );
    } while( end == false );

    //Clear the DMA transfer escape end flag
    R_PG_DMAC_ClearTransferEscapeEndFlag_C0();
}
```

5.8.15 R_PG_DMAC_ClearTransferEscapeEndFlag_C<*channel number*>

Definition

bool R_PG_DMAC_ClearTransferEscapeEndFlag_C<*channel number*> (void)
< *channel number* > : 0 to 3

Description

Clear the transfer escape end flag

Conditions for output

[Completion of a 1-block/repeat size transfer], [Source address extended repeat area overflow] or [Destination address extended repeat area overflow] is selected as the interrupt output source

Parameter

None

Return value

true	Clearing succeeded
false	Clearing failed

File for output

R_PG_DMAC_C <*channel number*>.c
<*channel number*>: 0 to 3

RPDL function

R_DMAC_Control

Details

- This function clears the transfer escape end flag.
- To get the transfer escape end flag, call R_PG_DMAC_GetTransferEscapeEndFlag_C<*channel number*>.

Example

A case where the setting is made as follows.

- DMAC0 is set to repeat transfer mode
- The transfer start trigger is interrupt
- [Completion of a 1-block/repeat size transfer] is selected for the interrupt output source
- The DMA interrupt priority level is 0

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

void func(void)
{
    bool end;

    //Set up DMAC0
    R_PG_DMAC_Set_C0();

    //Make DMAC0 be ready for the transfer start trigger
    R_PG_DMAC_Activate_C0();

    //Wait for the transfer escape end flag to become 1
    do{
        R_PG_DMAC_GetTransferEscapeEndFlag_C0( & end );
    } while( end == false );

    //Clear the DMA transfer escape end flag
    R_PG_DMAC_ClearTransferEscapeEndFlag_C0();
}
```

5.8.16 R_PG_DMAC_SetSrcAddress_C<*channel number*>

Definition	bool R_PG_DMAC_SetSrcAddress_C<*channel number*>(void * src_addr) < *channel number* > : 0 to 3
Description	Set the source address

Parameter

void * src_addr	The source address to be set

Return value

true	Setting was made correctly
false	Setting failed.

File for output: R_PG_DMAC_C<*channel number*>.c
<*channel number*>: 0 to 3

RPDL function: R_DMAC_Control

Details
- This function sets the source address.

Example

A case where the setting is made as follows.
- IRQ0 interrupt was selected as a transfer start trigger of DMAC0
- Dmac0IntFunc was specified as the DMA interrupt notification function name

```
#include "R_PG_default.h" //Include "R_PG_<PDG project name>.h" to use this function.
void func(void)
{
        //Set up DMAC0
        R_PG_DMAC_Set_C0();

        //Set IRQ0
        R_PG_ExtInterrupt_Set_IRQ0();

        // Make DMAC0 be ready for the transfer start trigger
        R_PG_DMAC_Activate_C0();
}
//DMA interrupt notification function
void Dmac0IntFunc (void)
{
       //Suspend the DMA transfer
       R_PG_DMAC_Suspend_C0();

        //Change the DMAC0 settings
        R_PG_DMAC_SetSrcAddress_C0( src_address );    //Source address
        R_PG_DMAC_SetDestAddress_C0( dest_address );    //Destination address
        R_PG_DMAC_SetTransferCount_C0( tr_count );    //Transfer counter

        // Make DMAC0 be ready for the transfer start trigger
        R_PG_DMAC_Activate_C0();
}
```

5.8.17 R_PG_DMAC_SetDestAddress_C<*channel number*>

Definition
bool R_PG_DMAC_SetDestAddress_C<*channel number*>(void * dest_addr)
< *channel number* > : 0 to 3

Description
Set the source address

Parameter

void * dest_addr	The destination address to be set

Return value

true	Setting was made correctly
false	Setting failed.

File for output
R_PG_DMAC_C<*channel number*>.c
<*channel number*>: 0 to 3

RPDL function
R_DMAC_Control

Details
- This function sets the destination address.

Example
A case where the setting is made as follows.
- IRQ0 interrupt was selected as a transfer start trigger of DMAC0
- Dmac0IntFunc was specified as the DMA interrupt notification function name

```
#include "R_PG_default.h" //Include "R_PG_<PDG project name>.h" to use this function.
void func(void)
{
	//Set up DMAC0
	R_PG_DMAC_Set_C0();
	//Set IRQ0
	R_PG_ExtInterrupt_Set_IRQ0();
	// Make DMAC0 be ready for the transfer start trigger
	R_PG_DMAC_Activate_C0();
}
//DMA interrupt notification function
void Dmac0IntFunc (void)
{
	//Suspend the DMA transfer
	R_PG_DMAC_Suspend_C0();
	//Set up the DMAC and continue
	R_PG_DMAC_SetSrcAddress_C0( src_address );	//Source address
	R_PG_DMAC_SetDestAddress_C0( dest_address );	//Destination address
	R_PG_DMAC_SetTransferCount_C0( tr_count );	//Transfer counter
	// Make DMAC0 be ready for the transfer start trigger
	R_PG_DMAC_Activate_C0();
}
```

5.8.18 R_PG_DMAC_SetAddressOffset_C<*channel number*>

Definition
: bool R_PG_DMAC_SetAddressOffset_C<*channel number*>(int32_t offset)
< *channel number* > : 0 to 3

Description
: Set the address offset

Conditions for output
: [Offset addition] is selected for [Source address update mode] or [Destination address update mode].

Parameter

int32_t offset	The offset value to be set

Return value

true	Setting was made correctly
false	Setting failed

File for output
: R_PG_DMAC_C<*channel number*>.c
<*channel number*>: 0 to 3

RPDL function
: R_DMAC_Control

Details
- This function sets the address offset.
- The range of the address offset value is from +FFFFFFh to -1000000h.

Example

A case where the setting is made as follows.

- IRQ0 interrupt was selected as a transfer start trigger of DMAC0
- Dmac0IntFunc was specified as the DMA interrupt notification function name
- [Offset addition] is selected.

```
#include "R_PG_default.h" //Include "R_PG_<PDG project name>.h" to use this function.
void func(void)
{
	//Set up DMAC0
	R_PG_DMAC_Set_C0();

	//Set IRQ0
	R_PG_ExtInterrupt_Set_IRQ0();

	// Make DMAC0 be ready for the transfer start trigger
	R_PG_DMAC_Activate_C0();
}
//DMA interrupt notification function
void Dmac0IntFunc (void)
{
	//Suspend the DMA transfer
	R_PG_DMAC_Suspend_C0();

	//Set up the DMAC and continue
	R_PG_DMAC_SetSrcAddress_C0( src_address );	//Source address
	R_PG_DMAC_SetDestAddress_C0( dest_address );	//Destination address
	R_PG_DMAC_SetTransferCount_C0( tr_count );	//Transfer counter
	R_PG_DMAC_SetAddressOffset_C0( offset );	//Address offset

	// Make DMAC0 be ready for the transfer start trigger
	R_PG_DMAC_Activate_C0();
}
```

5.8.19 R_PG_DMAC_SetExtendedRepeatSrc_C<*channel number*>

Definition
bool R_PG_DMAC_SetExtendedRepeatSrc_C<*channel number*>(uint32_t area)
< *channel number* > : 0 to 3

Description
Set the source address extended repeat value

Conditions for output
An extended repeat area is specified for the transfer source.

Parameter

uint32_t area	The source address extended repeat value to be set

Return value

true	Setting was made correctly
false	Setting failed

File for output
R_PG_DMAC_C<*channel number*>.c
<*channel number*>: 0 to 3

RPDL function
R_DMAC_Control

Details
- This function sets the source address extended repeat value.
- The value can be any power of 2, from 2^1 to 2^{27}.

Example
A case where the setting is made as follows.
- IRQ0 interrupt was selected as a transfer start trigger of DMAC0
- Dmac0IntFunc was specified as the DMA interrupt notification function name
- An extended repeat area is specified for the transfer source and destination.

```
#include "R_PG_default.h" //Include "R_PG_<PDG project name>.h" to use this function.
void func(void)
{
	//Set up DMAC0
	R_PG_DMAC_Set_C0();

	//Set IRQ0
	R_PG_ExtInterrupt_Set_IRQ0();

	// Make DMAC0 be ready for the transfer start trigger
	R_PG_DMAC_Activate_C0();
}
//DMA interrupt notification function
void Dmac0IntFunc (void)
{
	//Suspend the DMA transfer
	R_PG_DMAC_Suspend_C0();

	//Change the DMAC0 settings
	R_PG_DMAC_SetSrcAddress_C0( src_address );	//Source address
	R_PG_DMAC_SetDestAddress_C0( dest_address );	//Destination address
	R_PG_DMAC_SetTransferCount_C0( tr_count );	//Transfer counter
	R_PG_DMAC_SetExtendedRepeatSrc_C0( src_repeat );	//Source extended repeat size
	R_PG_DMAC_SetExtendedRepeatDest_C0( dest_repeat ); //Destination extended repeat size

	// Make DMAC0 be ready for the transfer start trigger
	R_PG_DMAC_Activate_C0();
}
```

5.8.20 R_PG_DMAC_SetExtendedRepeatDest_C<*channel number*>

Definition

bool R_PG_DMAC_SetExtendedRepeatDest_C<*channel number*>(uint32_t area)

< *channel number* > : 0 to 3

Description

Set the destination address extended repeat value

Conditions for output

An extended repeat area is specified for the transfer destination.

Parameter

uint32_t area	The destination address extended repeat value to be set

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_DMAC_C<*channel number*>.c

<*channel number*>: 0 to 3

RPDL function

R_DMAC_Control

Details

- This function sets the destination address extended repeat value.
- The value can be any power of 2, from 2^1 to 2^{27}.

Example

A case where the setting is made as follows.

- IRQ0 interrupt was selected as a transfer start trigger of DMAC0
- Dmac0IntFunc was specified as the DMA interrupt notification function name
- An extended repeat area is specified for the transfer source and destination.

```
#include "R_PG_default.h" //Include "R_PG_<PDG project name>.h" to use this function.
void func(void)
{
	//Set up DMAC0
	R_PG_DMAC_Set_C0();

	//Set IRQ0
	R_PG_ExtInterrupt_Set_IRQ0();

	// Make DMAC0 be ready for the transfer start trigger
	R_PG_DMAC_Activate_C0();
}
//DMA interrupt notification function
void Dmac0IntFunc (void)
{
	//Suspend the DMA transfer
	R_PG_DMAC_Suspend_C0();

	//Change the DMAC0 settings
	R_PG_DMAC_SetSrcAddress_C0( src_address );	//Source address
	R_PG_DMAC_SetDestAddress_C0( dest_address );	//Destination address
	R_PG_DMAC_SetTransferCount_C0( tr_count );	//Transfer counter
	R_PG_DMAC_SetExtendedRepeatSrc_C0( src_repeat );	//Source extended repeat size
	R_PG_DMAC_SetExtendedRepeatDest_C0( dest_repeat );	//Destination extended repeat size

	// Make DMAC0 be ready for the transfer start trigger
	R_PG_DMAC_Activate_C0();
}
```

5.8.21 R_PG_DMAC_StopModule_C<*channel number*>

Definition: bool R_PG_DMAC_StopModule_C<*channel number*> (void)
< *channel number* > : 0 to 3

Description: Stop the DMAC channel

Parameter: None

Return value:

true	Stopping succeeded.
false	Stopping failed.

File for output: R_PG_DMAC_C<*channel number*>.c
<*channel number*>: 0 to 3

RPDL function: R_DMAC_Destroy

Details:
- Stops the DMAC channel.
- If all DMAC channels and DTC are stopped, DMAC and DTC shall be module-stop state.
- If another peripheral is being used to trigger a DMA transfer, stop the trigger sources before calling this function.

Example: A case where the setting is made as follows.
- The software trigger was selected as a transfer start trigger of DMAC0
- Dmac0IntFunc was specified as the DMA interrupt notification function name

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

void func(void)
{
	//Set up DMAC0
	R_PG_DMAC_Set_C0();

	//Start the DMA transfer of DMAC0
	R_PG_DMAC_StartTransfer_C0();
}

//DMA interrupt notification function
void Dmac0IntFunc (void)
{
	//Stop the DMAC0
	R_PG_DMAC_StopModule_C0();
}
```

5.9 Data Transfer Controller (DTCa)

5.9.1 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:
- Releases DTC and DMAC from the module-stop state.
- Before calling other functions of DTC, call this function.
- 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 800h.
- The transfer setting of which the transfer start trigger is IRQ0 has been made.

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

//DTC vector table
#pragma address dtc_vector_table = 0x00000800
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();
}
```

5.9.2 R_PG_DTC_Set_<*trigger source*>

Definition bool R_PG_DTC_Set_<*trigger source*> (void)

< *trigger source* >

SWINT	Software interrupt
CMI0 to 3	CMT0 to 3 compare match interrupt
D0FIFO0	DMA transfer request 0
D1FIFO0	DMA transfer request 1
SPRI0 to 1	RSPI0 to 1 receive interrupt
SPTI0 to 1	RSPI0 to 1 transmit interrupt
IRQ0 to 7	External interrupts
S12ADI	S12AD scan end interrupt
S12GBADI	S12AD scan end interrupt for gropu B
S12ADI1	S12AD1 scan end interrupt
S12GBADI1	S12AD1 scan end interrupt for gropu B
TGIA0 to D0	MTU0 input capture/compare match A to D interrupt
TGIA1 or B1	MTU1 input capture/compare match A or B interrupt
TGIA2 or B2	MTU2 input capture/compare match A or B interrupt
TGIA3 to D3	MTU3 input capture/compare match A to D interrupt
TGIA4 to D4	MTU4 input capture/compare match A to D interrupt
TCIV4	MTU4 overflow/underflow interrupt
TGIU5 to W5	MTU5 input capture/compare match U to W interrupt
TGIA6 to D6	MTU6 input capture/compare match A to D interrupt
TGIA7 to D7	MTU7 input capture/compare match A to D interrupt
TCIV7	MTU7 overflow/underflow interrupt
CMP0 to 2, and 4 to 6	Comparator 0 to 2, and 4 to 6 detection interrupt
ICRXI0 to 1	RIIC0 to 1 receive data full interrupt
ICTXI0 to 1	RIIC0 to 1 transmit data empty interrupt
DMAC0I to 3I	DMACA0 to 3 interrupt
RXI0 to 3, and 12	SCI0 to 3, and 12 receive data full interrupt
TXI0 to 3, and 12	SCI0 to 3, and 12 transmit data empty interrupt
GTCIA0 or B0	GPT0 input capture/compare match A or B interrupt
GTCIC0	GPT0 compare match C/dead time error interrupt
GTCIE0	GPT0 compare match E interrupt
GTCIV0	GPT0 overflow/underflow interrupt
LOCOI0	External trigger/IWDTCLK cout function interrupt
GTCIA1 or B1	GPT1 input capture/compare match A or B interrupt
GTCIC1	GPT1 compare match C/dead time error interrupt
GTCIE1	GPT1 compare match E interrupt
GTCIV1	GPT1 overflow/underflow interrupt
GTCIA2 or B2	GPT2 input capture/compare match A or B interrupt

GTCIC2	GPT2 compare match C/dead time error interrupt
GTCIE2	GPT2 compare match E interrupt
GTCIV2	GPT2 overflow/underflow interrupt
GTCIA3 or B3	GPT3 input capture/compare match A or B interrupt
GTCIC3	GPT3 compare match C/dead time error interrupt
GTCIE3	GPT3 compare match E interrupt
GTCIV3	GPT3 overflow/underflow interrupt
GTCIA4 or B4	GPT4 input capture/compare match A or B interrupt
GTCIC4	GPT4 compare match C/dead time error interrupt
GTCIE4	GPT4 compare match E interrupt
GTCIV4	GPT4 overflow/underflow interrupt
LOCOI4	External trigger/IWDTCLK cout function interrupt
GTCIA5 or B5	GPT5 input capture/compare match A or B interrupt
GTCIC5	GPT5 compare match C/dead time error interrupt
GTCIE5	GPT5 compare match E interrupt
GTCIV5	GPT5 overflow/underflow interrupt
GTCIA6 or B6	GPT6 input capture/compare match A or B interrupt
GTCIC6	GPT6 compare match C/dead time error interrupt
GTCIE6	GPT6 compare match E interrupt
GTCIV6	GPT6 overflow/underflow interrupt
GTCIA7 or B7	GPT7 input capture/compare match A or B interrupt
GTCIC7	GPT7 compare match C/dead time error interrupt
GTCIE7	GPT7 compare match E interrupt
GTCIV7	GPT7 overflow/underflow interrupt

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.
- Call this function before configuring the peripherals that will be involved in the data transfer.

Example A case where the setting is made as follows.
- The DTC vector table address has been set to 800h.
- 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_<PDG project name>.h" to use this function.
#include "R_PG_default.h"

#pragma address dtc_vector_table = 0x00000800
uint32_t dtc_vector_table [256];    //DTC vector table

//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 IRQ
        R_PG_DTC_Set_IRQ0();
        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();
}
```

5.9.3 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 800h.
- 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_<PDG project name>.h" to use this function.
#include "R_PG_default.h"

//DTC vector table
#pragma address dtc_vector_table = 0x00000800
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();
}
```

5.9.4 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 enable the transfer.

Example A case where the setting is made as follows.
- The DTC vector table address has been set to 800h.
- The transfer setting of which the transfer start trigger is IRQ0 has been made.

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

//DTC vector table
#pragma address dtc_vector_table = 0x00000800
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();
}
```

5.9.5 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 storage area for the vector number of current data transfer (Valid when "* active" is 1)
bool * active	The address of 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_<PDG 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:
		}
	}
}
```

5.9.6 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.
- This function will also shut down the DMAC.

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

- The DTC vector table address has been set to 800h.
- 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_<PDG project name>.h" to use this function.
#include "R_PG_default.h"

//DTC vector table
#pragma address dtc_vector_table = 0x00000800
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();
}
```

5.10 I/O Ports

5.10.1 R_PG_IO_PORT_Set_P<*port number*>

Definition	bool R_PG_IO_PORT_Set_P<*port number*> (void) <*port number*>: 0 to 9, A to G
Description	Set up the I/O port
Conditions for output	When [Used as an I/O port] of one or more pins are specified in the port in the GUI. The function isn't generated to the port where it can't output.
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*>: 0 to 9, A to G
RPDL function	R_IO_PORT_Set

Details

- Selects the direction (input or output), and output type for pins for which [Used as an I/O port] was specified in the GUI.
- This function sets all pins for which [Used as I/O port] has been selected in a port.
- To set the output type to N-channel open-drain without changing the direction, use R_PG_IO_PORT_SetOpenDrain_P<*port number*><*pin number*> instead of this function.

Example

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

void func(void)
{
	//Configure I/O port pins that are not available.
	R_PG_IO_PORT_SetPortNotAvailable();

	//Set P0.
	R_PG_IO_PORT_Set_P0();
}
```

5.10.2 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*>: 0 to 9, A to G
<*pin number*>: 0 to 7

Description: Set up the I/O port pin

Conditions for output: When [Used as an I/O port] is specified in the GUI.
The function isn't generated to the port where it can't output.

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*>: 0 to 9, A to G

RPDL function: R_IO_PORT_Set

Details:
- Selects the direction (input or output), and output type for pins for which [Used as an I/O port] was specified in the GUI.
- The setting only applies to one pin.
- To set the output type to N-channel open-drain without changing the direction, use R_PG_IO_PORT_SetOpenDrain_P<*port number*><*pin number*> instead of this function.

Example:

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

void func(void)
{
	//Set P00.
	R_PG_IO_PORT_Set_P00();

	//Set P01.
	R_PG_IO_PORT_Set_P01();
}
```

5.10.3 R_PG_IO_PORT_Read_P<*port number*>

Definition
bool R_PG_IO_PORT_Read_P<*port number*> (uint8_t * data)
<*port number*>: 0 to 9, A to G

Description
Read data from Port Input Register

Conditions for output
When [Used as an I/O port] of one or more pins are specified in the port in the GUI.

Parameter

uint8_t * data	Destination for 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*>: 0 to 9, A to G

RPDL function
R_IO_PORT_Read

Details
- Reads Port Input Register to acquire the states of the pins. (Unit: Port)

Example

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

uint8_t data;

void func(void)
{
	//Acquire the states of P0 pins.
	R_PG_IO_PORT_Read_P0( &data );
}
```

5.10.4 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*>: 0 to 9, A to G
<*pin number*>: 0 to 7

Description

Read 1-bit data from Port Input Register

Conditions for output

When [Used as an I/O port] of one or more pins are specified in the port in the GUI, the function of all existing pins in the port is generated.

Parameter

uint8_t * data	Destination for 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*>: 0 to 9, A to G)

RPDL function

R_IO_PORT_Read

Details

- Reads Port Input Register to acquire the state of one pin.
- The value is stored in the lowest-order bit of *data.

Example

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

uint8_t data_p00, data_p01;

void func(void)
{
	//Acquire the state of pin P00.
	R_PG_IO_PORT_Read_P00( & data_p00);

	//Acquire the state of pin P01.
	R_PG_IO_PORT_Read_P01( & data_p01);
}
```

5.10.5 R_PG_IO_PORT_Write_P<*port number*>

Definition: bool R_PG_IO_PORT_Write_P<*port number*> (uint8_t data)
<*port number*>: 0 to 3, 7 to 9, A, B, D to G

Description: Write data to Port Output Data Register

Conditions for output: When [Used as an I/O port] of one or more pins are specified in the port in the GUI.
The function isn't generated to the port where it can't output.

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*>: 0 to 3, 7, 9, A to G

RPDL function: R_IO_PORT_Write

Details:
- Writes a value to Port Output Data Register. A value written to the register is output from the output port.

Example:

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

void func(void)
{
	//Set P0.
	R_PG_IO_PORT_Set_P0();

	//Output 0x03 from P0.
	R_PG_IO_PORT_Write_P0( 0x03 );
}
```

5.10.6 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*>: 0 to 3, 7 to 9, A, B, D to G <*pin number*>: 0 to 7
Description	Write 1-bit data to Port Output Data Register
Conditions for output	When [Used as an I/O port] is specified in the GUI. The function isn't generated to the port where it can't output.

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*>: 0 to 3, 7 to 9, A, B, D to G
RPDL function	R_IO_PORT_Write

Details

- Writes a value to Port Output 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_<PDG project name>.h" to use this function.
#include "R_PG_default.h"

void func(void)
{
	//Set P00.
	R_PG_IO_PORT_Set_P00();

	//Set P01.
	R_PG_IO_PORT_Set_P01();

	//Output low level from P00.
	R_PG_IO_PORT_Write_P00( 0x00 );

	//Output high level from P01.
	R_PG_IO_PORT_Write_P01( 0x01 );
}
```

5.10.7 R_PG_IO_PORT_SetPortNotAvailable

Definition bool R_PG_IO_PORT_SetPortNotAvailable (void)

Description Configure I/O port pins that are not available.

Parameter None

Return value

true	Setting was made correctly

File for output R_PG_IO_PORT.c

RPDL function R_IO_PORT_NotAvailable

Details
- All ports that are not available on smaller packages will be configured for CMOS-type low-level output.
- When using packages other than 144-pin or 64-pin, call this function first.

Example Refer to the example of R_PG_IO_PORT_Set_P<*port number*>.

5.10.8 R_PG_IO_PORT_SetOpenDrain_P*<port number><pin number>*

Definition: bool R_PG_IO_PORT_SetOpenDrain_P<*port number*><*pin number*> (void)
<*port number*>: 0, 2, 3, 8, 9, A, B, D, F and G
<*pin number*>: 0 to 7

Description: Set the output type of I/O port pin to N-channel open-drain.

Conditions for output: [Use as I/O port] is checked, [Output] is selected for the direction and [N-channel open-drain] is selected for the output type.

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*>: 0, 2, 3, 8, 9, A, B, D, F and G

RPDL function: R_IO_PORT_Set

Details:
- Sets the output type of one I/O port pin to N-channel open-drain.
- This function sets only an output type and does not changes the direction of I/O port.

Example:

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

void func(void)
{
	// Set the output type of P02 to N-channel open-drain
	R_PG_IO_PORT_SetOpenDrain_P02();
}
```

5.10.9 R_PG_IO_PORT_SetDriveHigh_DSCR*<register number>*_*<bit number>*

Definition
bool R_PG_IO_PORT_SetDriveHigh_DSCR*<register number>*_*<bit number>* (void)
*<register number>*_*<bit number>*: 1_1, 1_2, 1_3, 1_4, 1_5, 1_6, 1_7, 2_6 and 2_7

Description
Set the drive capacity to high.

Parameter
None

Return value

true	Setting was made correctly.
false	Setting failed.

File for output
R_PG_IO_PORT.c

RPDL function
R_IO_PORT_Set

Details
- Writes 1 to the bit in Driving Ability Control Register 1 or 2 (DSCR1 or DSCR2) that corresponds to *<register number>*_*<bit number>* to set drive capacity of corresponding ports to high.
- For the ports corresponding to the bits in DSCR1 and DSCR2, please refer to RX63T Group User’s Manual: Hardware.

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

void func(void)
{
	// Set the drive capacity of ports that corresponds to DSCR1 b3 to high
	R_PG_IO_PORT_SetDriveHigh_DSCR1_3();
}
```

5.11 Multi-Function Timer Pulse Unit 3 (MTU3)

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

Definition bool R_PG_Timer_Set_MTU_U*<unit number>*_*<channels>* (void)
<unit number>: 0
<channels>: C0 to C7
C3_C4, C6_C7(Complementary PWM mode or reset-synchronized PWM mode)

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 section 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 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 by R_PG_Timer_Set_MTU_U0_C3_C4. Chanells 6 and 7 are set up by R_PG_Timer_Set_MTU_U0_C6_C7.
- 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_<PDG project name>.h" to use this function.

void func(void)
{
	R_PG_Timer_Set_MTU_U0_C6();	//Set up the MTU1
	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_<PDG 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
	);
}
```

5.11.2 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_<PDG 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
}
```

5.11.3 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*>

5.11.4 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_<PDG 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
}
```

5.11.5 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, MTU6 and MTU7

uint16_t * counter_val	Destination for storage of the counter value

For MTU5

uint16_t * counter_u_val	Destination for storage of the counter U value
uint16_t * counter_v_val	Destination for storage of the counter V value
uint16_t * counter_w_val	Destination for 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_<PDG 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 );
}
```

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

<u>Definition</u>

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

<u>Description</u>

Set the MTU counter value

<u>Parameter</u>

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

<u>Return value</u>

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

<u>File for output</u>

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

<u>RPDL function</u>

R_MTU3_ControlChannel

<u>Details</u>

- Set the counter value of a MTU.

<u>Example</u>

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_<PDG 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
}
```

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

}
```

5.11.8 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_<PDG 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();
}
```

5.11.9 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 storage area for the general register A value
uint16_t* tgr_b_val	The address of storage area for the general register B value
uint16_t* tgr_c_val	The address of storage area for the general register C value
uint16_t* tgr_d_val	The address of storage area for the general register D value
uint16_t* tgr_e_val	The address of storage area for the general register E value
uint16_t* tgr_f_val	The address of storage area for the general register F value
uint16_t* tgr_u_val	The address of storage area for the general register U value
uint16_t* tgr_v_val	The address of storage area for the general register V value
uint16_t* tgr_w_val	The address of 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_<PDG 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
	);
}
```

5.11.10 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*>:

MTU0	: A, B, C, D, E or F
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_<PDG 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
}
```

5.11.11 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_<PDG 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 );
}
```

5.11.12 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_<PDG project name>.h" to use this function.

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

5.11.13 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_<PDG project name>.h" to use this function.

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

5.11.14 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*>

5.11.15 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_<PDG 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 );
}
```

5.11.16 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_<PDG project name>.h" to use this function.

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

5.12 Port Output Enable 3 (POE3)

5.12.1 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 and 4 pins, the POE pins used for high-impedance request signal input, and the output enable interrupt.
- The MTU 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 section Notes on Notification Functions.

A case where the setting is made as follows.

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

```
//Include "R_PG_<PDG 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
}
```

5.12.2 R_PG_POE_SetHiZ_<*Timer channels*>

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

<*Timer channels*>: MTU3467_GPT012, 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_<PDG 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
}
```

5.12.3 R_PG_POE_GetRequestFlagHiZ_<*Timer channels/flag*>

Definition
bool R_PG_POE_GetRequestFlagHiZ_MTU3467_GPT012 (bool * poe0)
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)
bool R_PG_POE_GetRequestFlagHiZ_OSTSTF (bool * oststf)

Description
Acquire the high-impedance request flags

Parameter

bool* poe0	The address of storage area for POE0# high-impedance request flags
bool* poe8	The address of storage area for POE8# high-impedance request flags
bool* poe10	The address of storage area for POE10# high-impedance request flags
bool* poe11	The address of storage area for POE11# high-impedance request flags
bool * oststf	The address of storage area for OSTST high-impedance flag

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 (POEnF). (n:0, 8, 10 and 11)
- 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_<PDG 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_MTU3467_GPT012( &poe0 );
	}while( ! poe0 );

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

	R_PG_POE_ClearFlag_MTU3467_GPT012();	//Clear high-impedance request flag
}
```

5.12.4 R_PG_POE_GetShortFlag_<*Timer channels*>

Definition bool R_PG_POE_GetShortFlag_<*Timer channels*> (bool * detected)

<*Timer channels*>: MTU3456_GPT012

Description Acquire the MTU output short flags

Parameter

bool* detected	The address of storage area for the output short flag (OSF1)

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 or GPT0, 1, 2 output short flags (OSF1).

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_<PDG 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_MTU3464_GOT012 (&detected);

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

                R_PG_POE_ClearFlag_MTU3467_GPT012();    // Clear the output short flag(OSF1)
        }
}
```

5.12.5 R_PG_POE_ClearFlag_<*Timer channels/flag*>

Definition
bool R_PG_POE_ClearFlag_<*Timer channels/flag*> (void)
<*Timer channels/flag*>: MTU3467_GPT012, MTU0, GPT0_1, GPT2_3, OSTSTF

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 / flag	Flags
MTU3, 4, 6, 7 GPT0, 1, 2	POE0 request flag (POE0F) MTU3,4, 6, 7, GPT0, 1, 2 output short flag (OSF1)
MTU0	POE8 request flag (POE8F)
GPT0_1	POE10 request flag (POE10F)
GPT2_3	POE11 request flag (POE11F)
OSTSTF	OSTST high-impedance flag

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

5.13 General PWM TImer (GPT)

5.13.1 R_PG_Timer_Set_GPT_U<*unit number*>

Definition
bool R_PG_Timer_Set_GPT_U<*unit number*>
<*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_<PDG 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
}
```

5.13.2 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	Halting succeeded.
false	Halting failed.

File for output — R_PG_Timer_TPU_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*>.

5.13.3 R_PG_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*>.

5.13.4 R_PG_Timer_SynchronouslyStartCount_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_<PDG 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 );
}
```

5.13.5 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_<PDG 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();
}
```

5.13.6 R_PG_Timer_SynchronouslyHaltCout_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*>.

5.13.7 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*>.

5.13.8 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 storage area for compare capture register A value
uint16_t * gtccr_b_val	The address of storage area for compare capture register B value
uint16_t * gtccr_c_val	The address of storage area for compare capture register C value
uint16_t * gtccr_d_val	The address of storage area for compare capture register D value
uint16_t * gtccr_e_val	The address of storage area for compare capture register E value
uint16_t * gtccr_f_val	The address of 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_<PDG 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)
        );
}
```

5.13.9 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*>.

5.13.10 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 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_<PDG 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 );
}
```

5.13.11 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_<PDG 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 );
}
```

5.13.12 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_<PDG 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 );
}
```

5.13.13 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_<PDG 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 );
}
```

5.13.14 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_<PDG 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 );
}
```

5.13.15 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_<PDG 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)
	);
}
```

5.13.16 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_<PDG 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)
	);
}
```

5.13.17 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_<PDG 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
	);
}
```

5.13.18 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_<PDG 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 );
}
```

5.13.19 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 storage area for the compare match/input capture A flag
bool * cm_ic_b	The address of storage area for the compare match/input capture B flag
bool * cm_c	The address of storage area for the compare match/input capture C flag
bool * cm_d	The address of storage area for the compare match/input capture D flag
bool * cm_e	The address of storage area for the compare match/input capture E flag
bool * cm_f	The address of storage area for the compare match/input capture F flag
bool * ov	The address of storage area for the overflow flag
bool * un	The address of storage area for the underflow flag
bool * dt_error	The address of 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_<PDG 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 )
    }
}
```

5.13.20 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 storage area for the frequency-divided LOCO clock rise interrupt request flag
bool * loco_deviation	The address of storage area for the fLOCO count value deviation exceedance interrupt request flag
bool * loco_ov	The address of storage area for the LCNT overflow interrupt request flag
bool * ext_rising	The address of storage area for the external trigger rising input interrupt request flag
bool * ext_falling	The address of 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_<PDG 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)
        );
}
```

5.13.21 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 storage area for the count start bit (0 : Count operation is performed 1 : Count operation is stopped)
bool * up	The address of 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_<PDG 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
	);
}
```

5.13.22 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_<PDG 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
        );
}
```

5.13.23 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_<PDG 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
	);
}
```

5.13.24 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_<PDG 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
}
```

5.13.25 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_<PDG 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 );
}
```

5.13.26 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_<PDG 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 );
}
```

5.13.27 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_<PDG 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 );
}
```

5.13.28 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_<PDG 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();
}
```

5.13.29 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_<PDG 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();
}
```

5.13.30 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_<PDG 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();
}
```

5.13.31 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_<PDG 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();
}
```

5.13.32 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 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_<PDG 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 );
}
```

5.13.33 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 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_<PDG 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 );
}
```

5.13.34 R_PG_Timer_GetCounterResultValue_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 to LCNT15) value.

Example

```
//Include “R_PG_<PDG 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 );
}
```

5.13.35 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_<PDG 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
	);
}
```

5.13.36 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);
}
```

5.13.37 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_DisbleEdgeDelay_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);
}
```

5.13.38 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);
}
```

5.13.39 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_<PDG 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();
}
```

5.14 Compare Match Timer (CMT)

5.14.1 R_PG_Timer_Set_CMT_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_Set_CMT_U<*unit number*>_C<*channel number*> (void)
<*unit number*>: 0 or 1
<*channel number*>: 0 to 3

Description

Set up the CMT

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 and makes initial settings.
- R_PG_Timer_StartCount_CMT_U<*unit number*>_C<*channel number*> can be used to start the count operation.
- Function R_PG_Clock_Set must be called before any use of this function.
- 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 this chapter end, 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_<PDG project name>.h" to use this function.

void func(void)
{
        R_PG_Clock_Set(); //The clock-generation circuit has to be set first.
        R_PG_Timer_Set_CMT_U0_C0(); //Set up the CMT0
        R_PG_Timer_StartCount_CMT_U0_C0(); //Start the count operation
}

void Cmt0IntFunc(void)
{
        R_PG_Timer_HaltCount_CMT_U0_C0(); //Halt the CMT0 count operation

        func_cmt0();    //Processing in response to a compare match interrupt

        R_PG_Timer_StartCount_CMT_U0_C0(); //Resume the CMT0 count operation
}
```

5.14.2 R_PG_Timer_StartCount_CMT_U<*unit number*>_C<*channel number*>

Definition: bool R_PG_Timer_StartCount_CMT_U<*unit number*>_C<*channel number*> (void)
<*unit number*>: 0 or 1
<*channel number*>: 0 to 3

Description: Start or resume the CMT count operation

Parameter: None

Return value:

True	Starting or resuming count succeeded.
False	Starting or 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:
- Starts counting by a CMT.
- Resumes counting by a CMT that was halted by R_PG_Timer_HaltCount_CMT_U<*unit number*>_C<*channel number*>.

Example: Refer to the example of R_PG_Timer_Set_CMT_U<*unit number*>_C<*channel number*>

5.14.3 R_PG_Timer_HaltCount_CMT_U<*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 make the CMT resume counting, call the following function.
  R_PG_Timer_StartCount_CMT_U<*unit number*>_C<*channel number*>

Example Refer to the example of R_PG_Timer_Set_CMT_U<*unit number*>_C<*channel number*>

5.14.4 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 * data)
<*unit number*>: 0 or 1
<*channel number*>: 0 to 3

Description
: Acquire the CMT counter value

Parameter

uint16_t * data	Destination for 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_<PDG project name>.h" to use this function.

uint16_t data;

void func1(void)
{
	R_PG_Clock_Set(); //The clock-generation circuit has to be set first.
	R_PG_Timer_Set_CMT_U0_C0(); //Set up the CMT0
	R_PG_Timer_StartCount_CMT_U0_C0(); //Start the count operation
}

void func2(void)
{
	//Acquire the value of a CMT0 counter
	R_PG_Timer_GetCounterValue_CMT_U0_C0( &data );
}
```

5.14.5 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 data)
<*unit number*>: 0 or 1
<*channel number*>: 0 to 3

Description

Set the CMT counter value

Parameter

uint16_t data	Value to be set 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_<PDG project name>.h" to use this function.

void func1(void)
{
	R_PG_Clock_Set(); //The clock-generation circuit has to be set first.
	R_PG_Timer_Set_CMT_U0_C0(); //Set up the CMT0
	R_PG_Timer_StartCount_CMT_U0_C0(); //Start the count operation
}

void func2(void)
{
	//Set the value of a CMT0 counter
	R_PG_Timer_SetCounterValue_CMT_U0_C0( 0 );
}
```

5.14.6 R_PG_Timer_SetConstantRegister_CMT_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_SetConstantRegister_CMT_U<*unit number*>_C<*channel number*>
(uint16_t constant_val)
<*unit number*>: 0 or 1
<*channel number*>: 0 to 3

Description

Set the CMT constant register value

Parameter

uint16_t constant_val	Destination for storage of the constant register value.

Return value

true	Setting was made correctly.
false	Setting 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 CMT constant register value.

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_<PDG project name>.h" to use this function.

void func1(void)
{
	R_PG_Clock_Set(); //The clock-generation circuit has to be set first.
	R_PG_Timer_Set_CMT_U0_C0(); //Set up the CMT0
	R_PG_Timer_StartCount_CMT_U0_C0(); //Start the count operation
}

void func2(void)
{
	//Set the CMT constant register value
	R_PG_Timer_SetConstantRegister_CMT_U0_C0( 0xabcd );
}
```

5.14.7 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_<PDG project name>.h" to use this function.

void func(void)
{
	R_PG_Clock_Set(); //The clock-generation circuit has to be set first.
	R_PG_Timer_Set_CMT_U0_C0(); //Set up the CMT0
	R_PG_Timer_StartCount_CMT_U0_C0(); //Start the count operation
}

void Cmt0IntFunc(void)
{
	func_cmt0(); //Processing in response to a compare match interrupt

	//Stop the CMT unit 0
	R_PG_Timer_StopModule_CMT_U0();
}
```

5.15 Watchdog Timer (WDTA)

5.15.1 R_PG_Timer_Start_WDT

Definition: bool R_PG_Timer_Start_WDT (void)

Description: Set up the WDT and start the count operation

Conditions for output: Register start mode is selected.
(This function is not output if auto-start mode is selected. A macro for setting option function select registers is output to R_PG_MCU_OFS.c.)

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Timer_ WDT.c

RPDL function: R_WDT_Set

Details:
- Makes initial settings of WDT and starts the count operation.

Example: A case where the setting is made as follows.
- Start mode: Register start mode

```
//Include “R_PG_<PDG 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
}
```

5.15.2 R_PG_Timer_RefreshCounter_WDT

Definition bool R_PG_Timer_RefreshCounter_WDT(void)

Description Refresh 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

- Refresh the counter of WDT

Example

```
//Include “R_PG_<PDG 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_RefreshCounter_WDT();      //Refresh the WDT counter
}
```

5.15.3 R_PG_Timer_GetStatus_WDT

Definition bool R_PG_Timer_GetStatus_WDT(uint16_t * counter_val, bool * undf, bool * ref_err)

Description Acquires the status flag and count value of WDT

Parameter

uint16_t * counter_val	The address of storage area for the counter value
bool * undf	The address of storage area for the underflow flag
bool * ref_err	The address of storage area for the refresh error 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 status flag and count value of WDT.

Example

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

uint16_t counter_val;
bool undf;
bool ref_err;

void func(void)
{
	//Acquires the status flag and count value of WDT
	R_PG_Timer_GetStatus_WDT(&counter_val, &undf, &ref_err);
}
```

5.16 Independent Watchdog Timer (IWDTa)

5.16.1 R_PG_Timer_Start_IWDT

Definition bool R_PG_Timer_Start_IWDT (void)

Description Sets up the IWDT and starts its timer

Conditions for output Register start mode is selected.
(This function is not output if auto-start mode is selected. A macro for setting option function select registers is output to R_PG_MCU_OFS.c.)

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
- This function sets up the IWDT and starts its counter.
- Before calling this function, call R_PG_Clock_Set to set the clock.

Example

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

void func(void)
{
	//Set up the clocks
	R_PG_Clock_Set();

	//Sets up the IWDT and starts its timer
	R_PG_Timer_Start_IWDT();
}
```

5.16.2 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
- After starting the count operation, call this function to clear the counter before the counter underflow.

Example

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

void func1(void)
{
	//Set up the clocks
	R_PG_Clock_Set();

	//Sets up the IWDT and starts its timer
	R_PG_Timer_Start_IWDT();
}

void func2(void)
{
	//Refresh the counter
	R_PG_Timer_RefreshCounter_IWDT();
}
```

5.16.3 R_PG_Timer_GetStatus_IWDT

Definition bool R_PG_Timer_GetStatus_IWDT(uint16_t * counter_val, bool * undf, bool * ref_err)

Description Acquires the status flag and count value of IWDT

Parameter

uint16_t * counter_val	The address of storage area for the IWDT counter value
bool * undf	The address of storage area for the underflow flag
bool * ref_err	The address of storage area for the refresh error flag

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 status flag and counter value.
- The underflow flag shall be cleared in this function.

Example

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

uint16_t counter_val;
bool undf;
bool ref_err;

void func(void)
{
	//Acquires the IWDT status flag and counter value
	R_PG_Timer_GetStatus_IWDT(&counter_val, &undf, &ref_err);
}
```

5.17 Serial Communications Interface (SCIc, SCId)

5.17.1 R_PG_SCI_Set_C<*channel number*>

Definition bool R_PG_SCI_Set_C<*channel number*> (void)

<*channel number*>: 0, 1, 2, 3 and 12

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, 1, 2, 3 and 12

RPDL function R_SCI_Create, R_SCI_Set

Details

- Releases a SCI channel from the module-stop state, makes initial settings.
- 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 this chapter end, Notes on Notification Functions.

Example SCI0 has been set in the GUI.

```
//Include "R_PG_<PDG 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.
}
```

5.17.2 R_PG_SCI_SendTargetStationID_C<*channel number*>

Definition
bool R_PG_SCI_SendTargetStationID_C<*channel number*> (uint8_t id)
<*channel number*>: 0, 1, 2, 3 and 12

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, 1, 2, 3 and 12

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 SCI0 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_<PDG project name>.h" to use this function.
#include "R_PG_default.h"

uint8_t data[] = “ABCDEFGHIJ”;

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_SendTargetStationID_C0( 5 );	//Send ID code (ID:5)
	R_PG_SCI_SendAllData_C0( data, 10 );	//Send data
}
```

5.17.3 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, 1, 2, 3 and 12

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, 1, 2, 3 and 12

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 this chapter end, 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_<PDG 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)
{
    R_PG_SCI_StopModule_C0();               //Shut down the SCI0
}
```

5.17.4 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, 1, 2, 3 and 12

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, 1, 2, 3 and 12

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_<PDG 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
}
```

5.17.5 R_PG_SCI_I2CMode_Send_C<*channel number*>

Definition
bool R_PG_SCI_I2CMode_Send_C<*channel number*>
(bool addr_10bit, uint16_t slave, uint8_t * data, uint16_t count)
<*channel number*>: 0, 1, 2, 3 and 12

Description
Transmit data by simple I^{2}C bus interface

Conditions for output
- Simple I^{2}C bus interface is selected for “Mode”.

Parameter

bool addr_10bit	Slave address format (1: 10bit 0: 7bit)
uint16_t slave	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	When [Wait at the transmission function until all data has been transmitted] was selected for data transmission method, the operation completed OK. When except [Wait at the transmission function until all data has been transmitted] is selected for data transmission method, return value is always “true”.
false	When [Wait at the transmission function until all data has been transmitted] was selected for data transmission method, an error was detected.

File for output
R_PG_SCI_C<*channel number*>.c
<*channel number*>: 0, 1, 2, 3 and 12

RPDL function
R_SCI_IIC_Write

Details
- Transmit data by simple I^{2}C bus interface.

Example
[SCI0]
Mode: Simple I2C mode
Data transmission method: Notify the transmission completion of all data by function call

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

uint8_t data_tr[] = “ABCDEFGHIJ”;
uint16_t tr_count;

void func(void)
{
	R_PG_Clock_Set();		//The clock-generation circuit has to be set first.
	R_PG_SCI_Set_C0();		//Set up SCI0.

	//Transmit data by simple I2C bus interface
	R_PG_SCI_I2CMode_Send_C0(0, 0x0006, data_tr, 10);
}

void Sci0TrFunc(void)
{
	//Acquire the number of transmitted data
	R_PG_SCI_GetSentDataCount_C0(&tr_count);
}
```

5.17.6 R_PG_SCI_I2CMode_SendWithoutStop_C<*channel number*>

Definition
bool R_PG_SCI_I2CMode_SendWithoutStop_C<*channel number*>
(bool addr_10bit, uint16_t slave, uint8_t * data, uint16_t count)
<*channel number*>: 0, 1, 2, 3 and 12

Description
Transmit data by simple I^2C bus interface (no stop condition)

Conditions for output
- Simple I^2C bus interface is selected for “Mode”.

Parameter

bool addr_10bit	Slave address format (1: 10bit 0: 7bit)
uint16_t slave	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	When [Wait at the transmission function until all data has been transmitted] was selected for data transmission method, the operation completed OK. When except [Wait at the transmission function until all data has been transmitted] is selected for data transmission method, return value is always “true”.
false	When [Wait at the transmission function until all data has been transmitted] was selected for data transmission method, an error was detected.

File for output
R_PG_SCI_C<*channel number*>.c
<*channel number*>: 0, 1, 2, 3 and 12

RPDL function
R_SCI_IIC_Write

Details
- Transmit data by simple I^2C bus interface (no stop condition).

Example
[SCI0]
Mode: Simple I2C mode
Data transmission method: Notify the transmission completion of all data by function call

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

uint8_t data_tr[10];
uint8_t data_re[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.

	//Transmit data by simple I2C bus interface (no stop condition)
	R_PG_SCI_I2CMode_SendWithoutStop_C0(0, 0x0006, data_tr, 10);
}

void Sci0TrFunc(void)
{
	//Receive data by simple I2C bus interface (RE-START condition)
	R_PG_SCI_I2CMode_RestartReceive_C0(0, 0x0006, data_re, 10);
}
```

5.17.7 R_PG_SCI_I2CMode_GenerateStopCondition_C<*channel number*>

Definition: bool R_PG_SCI_I2CMode_GenerateStopCondition_C<*channel number*> (void)
<*channel number*>: 0, 1, 2, 3 and 12

Description: Generate a stop condition

Conditions for output:
- Simple I^2C bus interface is selected for “Mode”.

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, 1, 2, 3 and 12

RPDL function: R_SCI_Control

Details:
- This function generates a stop condition.

Example:
[SCI0]
Mode: Simple I2C mode
Data transmission method: Transfer the transmitted serial data by DMAC

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

uint8_t data_tr[]=”ABCDEFGHIJ”;

void func(void)
{
	R_PG_Clock_Set();		//The clock-generation circuit has to be set first.

	//Set up a DMAC channel
	R_PG_DMAC_Set_C0();

	//Set the source address
	R_PG_DMAC_SetSrcAddress_C0(data_tr);

	//Make the DMAC be ready for the start trigger
	R_PG_DMAC_Activate_C0();

	//Set up a SCI channel
	R_PG_SCI_Set_C0();

	//Transmit data by simple I2C bus interface
	R_PG_SCI_I2CMode_Send_C0(0, 0x0006, data_tr, 10);
}

void Dmac0IntFunc(void)
{
	//Generate a stop condition
	R_PG_SCI_I2CMode_GenerateStopCondition_C0();
}
```

5.17.8 R_PG_SCI_I2CMode_Receive_C<*channel number*>

Definition
bool R_PG_SCI_I2CMode_Receive_C<*channel number*>
(bool addr_10bit, uint16_t slave, uint8_t * data, uint16_t count)
<*channel number*>: 0, 1, 2, 3 and 12

Description
Receive data by simple I^{2}C bus interface

Conditions for output
- Simple I^{2}C bus interface is selected for “Mode”.

Parameter

bool addr_10bit	Slave address format (1: 10bit 0: 7bit)
uint16_t slave	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	When [Wait at the reception function until all data has been received] was selected for data reception method, the operation completed OK. When except [Wait at the reception function until all data has been received] is selected for data reception method, return value is always “true”.
false	When [Wait at the reception function until all data has been received] was selected for data reception method, an error was detected.

File for output
R_PG_SCI_C<*channel number*>.c
<*channel number*>: 0, 1, 2, 3 and 12

RPDL function
R_SCI_IIC_Read

Details
- This function receives data by simple I^{2}C bus interface.

Example
[SCI0]
Mode: Simple I2C mode
Function selection: Transmission and reception

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

uint8_t data_re[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.

	//Receive data by simple I2C bus interface
	R_PG_SCI_I2CMode_Receive_C0(0, 0x0006, data_re, 10);
}
```

5.17.9 R_PG_SCI_I2CMode_RestartReceive_C<*channel number*>

Definition
bool R_PG_SCI_I2CMode_RestartReceive_C<*channel number*>
(bool addr_10bit, uint16_t slave, uint8_t * data, uint16_t count)
<*channel number*>: 0, 1, 2, 3 and 12

Description
Receive data by simple I^2C bus interface (RE-START condition)

Conditions for output
- Simple I^2C bus interface is selected for “Mode”.

Parameter

bool addr_10bit	Slave address format (1: 10bit 0: 7bit)
uint16_t slave	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	When [Wait at the reception function until all data has been received] was selected for data reception method, the operation completed OK. When except [Wait at the reception function until all data has been received] is selected for data reception method, return value is always “true”.
false	When [Wait at the reception function until all data has been received] was selected for data reception method, an error was detected.

File for output
R_PG_SCI_C<*channel number*>.c <*channel number*>: 0, 1, 2, 3 and 12

RPDL function
R_SCI_IIC_Read

Details
- This function receives data by simple I^2C bus interface. (RE-START condition)

Example
[SCI0]
Mode: Simple I2C mode

```
//Include "R_PG_<PDG project name>.h" to use this function.
#include "R_PG_default.h"
uint8_t data_re[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.
    //Transmit data by simple I2C bus interface (no stop condition)
    R_PG_SCI_I2CMode_SendWithoutStop_C0(
        1,                  //10 bit address format
        0x0006,             //Slave address
        PDL_NO_PTR,         //The start address of the data to be sent
        PDL_NO_DATA         //The number of the data to be sent
    );
    //Receive data by simple I2C bus interface (RE-START condition)
    R_PG_SCI_I2CMode_RestartReceive_C0(
        0,          //7 bit address format
        0x00f0,     //Slave address
        data_re,    //The start address of the storage area for the expected data.
        10          //The number of the data to be received.
    );
}
```

5.17.10 R_PG_SCI_I2CMode_ReceiveLast_C<*channel number*>

Definition: bool R_PG_SCI_I2CMode_ReceiveLast_C<*channel number*> (uint8_t * data)
<*channel number*>: 0, 1, 2, 3 and 12

Description: Making reception complete in simple I^{2}C bus interface

Conditions for output:
- Simple I^{2}C bus interface is selected for “Mode”.
- “Transfer the received serial data by DMAC” or “Transfer the received serial data by DTC” is selected for data reception method.

Parameter

uint8_t * data	The start address of the storage area for the expected data.

Return value

true	Setting was made correctly
false	Setting failed

File for output: R_PG_SCI_C<*channel number*>.c
<*channel number*>: 0, 1, 2, 3 and 12

RPDL function: R_SCI_IIC_ReadLastByte

Details:
- After received data is transferred by the DMAC or DTC in simple I^{2}C mode, this function must be called to complete the reception.
- This function must be called from a DMA interrupt notification function or receive end notification function.

Example:
[SCI0]
Mode: Simple I2C mode
Data reception method: Transfer the received serial data by DMAC
[DMAC0]
Transfer request source: RXI0 (SCI0 receive data full interrupt)
Transfer mode: Normal transfer mode
Length of a single data: 1 byte
Number of times: 4
Start address: 8a005h
Notify DMA interrupt (DMACIn)
[DMAC1]
Transfer request source: TXI0 (SCI0 transmit data empty interrupt)
Transfer mode: Normal transfer mode
Length of a single data: 1 byte
Number of times: 3
Source address update mode: Fixed
Start address: 8a003h

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

uint8_t data_re[5];
uint8_t dummy_data=0xFF;

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_DMAC_Set_C0();    //Set up DMAC.
        R_PG_DMAC_Set_C1();    //Set up DMAC.
        R_PG_DMAC_SetDestAddress_C0(data_re);    //Set the destination address.
        R_PG_DMAC_SetSrcAddress_C1(&dummy_data);    //Set the source address.
        R_PG_DMAC_Active_C0();    //Make the DMAC be ready for the start trigger.
        R_PG_DMAC_Active_C1();    //Make the DMAC be ready for the start trigger.

        //Receive data by simple I2C bus interface.
        R_PG_SCI_I2CMode_Receive_C0(0, 0x0006, PDL_NO_PTR, 0);
}

void Dmac0IntFunc(void)
{
        //Making reception complete in simple I2C bus interface.
        R_PG_SCI_I2CMode_ReceiveLast_C0(&data_re[4]);
}
```

5.17.11 R_PG_SCI_I2CMode_GetEvent_C<*channel number*>

Definition: bool R_PG_SCI_I2CMode_GetEvent_C<*channel number*> (bool * nack)

<*channel number*>: 0, 1, 2, 3 and 12

Description: Get the detected event in the simple I^2C mode

Conditions for output: Simple I^2C bus interface is selected for “Mode”.

Parameter

bool * nack	The address of the storage area for a NACK detection flag.

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output: R_PG_SCI_C<*channel number*>.c

<*channel number*>: 0, 1, 2, 3 and 12

RPDL function: R_SCI_GetStatus

Details:

- This function acquires ACK Reception Data Flag in the simple I^2C mode.

Example: [SCI0]

Mode:Simple I^2C mode

Data transmission method:Notify the transmission completion of all data by function call

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

uint8_t data_tr[]=”ABCDEFGHIJ”;
bool nack;

void func(void)
{
	R_PG_Clock_Set();		//The clock-generation circuit has to be set first.
	R_PG_SCI_Set_C0();		//Set up SCI0.

	//Transmit data by simple I2C bus interface
	R_PG_SCI_I2CMode_Send_C0(0, 0x0006, data_tr, 10);
}

void Sci0TrFunc(void)
{
	//Get the detected event in the simple I2C mode
	R_PG_SCI_I2CMode_GetEvent_C0(&nack);
}
```

5.17.12 R_PG_SCI_SPIMode_Transfer_C<*channel number*>

Definition	bool R_PG_SCI_SPIMode_Transfer_C<*channel number*> (uint8_t * tx_start, uint8_t * rx_start, uint16_t count) <*channel number*>: 0, 1, 2, 3 and 12
Description	Transmit data by simple SPI mode
Conditions for output	• Simple SPI mode is selected for “Mode”.

Parameter

uint8_t * tx_start	The start address of the data to be transmitted.
uint8_t * rx_start	The start address of the storage area for the expected data.
uint16_t count	The number of the data to be transferred.

Return value

true	When [Wait at the transmission/reception function until all data has been transmitted/received] was selected for data transmission/reception method, the operation completed OK. When except [Wait at the transmission/reception function until all data has been transmitted/received] is selected for data transmission/reception method, return value is always “true”.
false	When [Wait at the transmission/reception function until all data has been transmitted/received] was selected for data transmission/reception method, an error was detected.

File for output	R_PG_SCI_C<*channel number*>.c <*channel number*>: 0, 1, 2, 3 and 12
RPDL function	R_SCI_SPI_Transfer
Details	• This function transmits data by simple SPI mode.

Example

[SCI0]

Mode: Simple SPI mode

Function selection: Transmission and reception

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

uint8_t data_tr[10];
uint8_t data_re[10];

void func1(void)
{
	R_PG_Clock_Set();		//The clock-generation circuit has to be set first.
	R_PG_SCI_Set_C0();		//Set up SCI0.
}

void func2(void)
{
	//Transmit data by simple SPI mode
	R_PG_SCI_SPIMode_Transfer_C0(data_tr, data_re, 10);
}
```

5.17.13 R_PG_SCI_SPIMode_GetErrorFlag_C<*channel number*>

Definition: bool R_PG_SCI_SPIMode_GetErrorFlag_C<*channel number*> (bool * overrun)
<*channel number*>: 0, 1, 2, 3 and 12

Description: Get the serial reception error flag in the simple SPI mode

Conditions for output: Simple SPI mode is selected for “Mode”.

Parameter

bool * overrun	The address of the storage area for the overrun error flag.

Return value

true	Acquisition of the flag succeeded
false	Acquisition of the flag failed

File for output: R_PG_SCI_C<*channel number*>.c
<*channel number*>: 0, 1, 2, 3 and 12

RPDL function: R_SCI_GetStatus

Details
- This function acquires the serial reception error flag in the simple SPI mode.
- Specify 0 for a flag that is not required.
- The flags of detected error will be set to 1.

Example
[SCI0]
Mode: Simple SPI mode
Function selection: Transmission and reception
Notify receive error detection by function call

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

uint8_t tx_data[4];
uint8_t rx_data[4];
bool overrun;

void func(void)
{
	R_PG_Clock_Set();		//The clock-generation circuit has to be set first.
	R_PG_SCI_Set_C0();		//Set up SCI0.

	//Transmit data by simple SPI mode
	R_PG_SCI_SPIMode_Transfer_C0(tx_data, rx_data, 4);
}

void Sci0ErFunc(void)
{
	//Get the serial reception error flag in the simple SPI mode
	R_PG_SCI_SPIMode_GetErrorFlag_C0(&overrun);
}
```

5.17.14 R_PG_SCI_GetSentDataCount_C<*channel number*>

Definition	bool R_PG_SCI_GetSentDataCount_C<*channel number*> (uint16_t * count) <*channel number*>: 0, 1, 2, 3 and 12
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, 1, 2, 3 and 12
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_<PDG 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.
}

//The transmit end notification function that called when all bytes have been sent
void Sci0TrFunc(void)
{
        R_PG_SCI_StopModule_C0();    //Shut down the SCI0
}

//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
        }
}
```

5.17.15 R_PG_SCI_ReceiveStationID_C<*channel number*>

Definition: bool R_PG_SCI_ReceiveStationID_C<*channel number*> (void)
<*channel number*>: 0, 1, 2, 3 and 12

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, 1, 2, 3 and 12

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_<PDG 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_ReceiveAllData_C0( data, 10 );		//Start receiving
}
```

5.17.16 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, 1, 2, 3 and 12

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, 1, 2, 3 and 12

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 this chapter end, 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_<PDG 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)
{
	R_PG_SCI_StopModule_C0();	//Shut down the SCI0
}
```

5.17.17 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, 1, 2, 3 and 12

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, 1, 2, 3 and 12

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_<PDG 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
}
```

5.17.18 R_PG_SCI_ControlClockOutput_C<*channel number*>

Definition	bool R_PG_SCI_ControlClockOutput_C<*channel number*> (bool output_enable) <*channel number*>: 0, 1, 2, 3 and 12
Description	Control the output from the SCKn pin (n: 0, 1, 2, 3 and 12)
Conditions for output	• “Smart card interface mode” is selected for mode. • “Enable (GSM mode)” is selected for GSM mode. • “Output fixed high” or “Output fixed low” is selected for SCKn pin function.

Parameter

bool output_enable	Output from the SCKn pin (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, 1, 2, 3 and 12
RPDL function	R_SCI_Control
Details	• This function controls the clock output from the SCKn pin.

Example

[SCI0]

Mode: Smart card interface mode

GSM mode: Enable

SCKn pin function: Output fixed high

```
//Include "R_PG_<PDG 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.

        //Control the output from the SCKn pin
        R_PG_SCI_ControlClockOutput_C0( 1 );
}
```

5.17.19 R_PG_SCI_StopCommunication_C<*channel number*>

Definition
R_PG_SCI_StopCommunication_C<*channel number*> (void)
<*channel number*>: 0, 1, 2, 3 and 12

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, 1, 2, 3 and 12

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
Refer to the example of R_PG_SCI_GetSentDataCount_C<*channel number*>

5.17.20 R_PG_SCI_GetReceivedDataCount_C<*channel number*>

Definition	bool R_PG_SCI_GetReceivedDataCount_C<*channel number*> (uint16_t * count) <*channel number*>: 0, 1, 2, 3 and 12
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, 1, 2, 3 and 12

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_<PDG 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.
}

//The receive end notification function that called when all bytes have been received.
void Sci0ReFunc(void)
{
	R_PG_SCI_StopModule_C0();	//Shut down the SCI0
}

//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_StopCommunication_C0();		//Terminate the reception
	}
}
```

5.17.21 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, 1, 2, 3 and 12

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 storage area for the parity error flag
bool * framing	The address of storage area for the framing error flag
bool * overrun	The address of 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, 1, 2, 3 and 12

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_<PDG 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, 1);	//Receive 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 );
}
```

5.17.22 R_PG_SCI_ClearReceptionErrorFlag_C<*channel number*>

Definition: bool R_PG_SCI_ClearReceptionErrorFlag_C<*channel number*> (void)
<*channel number*>: 0, 1, 2, 3 and 12

Description: Clear the serial reception error flag

Conditions for output:
- “Asynchronous mode”, “Clock synchronous mode” or “Smart card interface mode” is selected for mode.
- “Reception” or “Transmission and reception” is selected for function selection.

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, 1, 2, 3 and 12

RPDL function: R_SCI_Control

Details:
- This function clears the serial reception error flag.

Example:
Mode: Asynchronous mode
Function selection: Reception
Data reception method: Notify the reception completion of all data by function call

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

uint8_t data_re[10];
bool parity, framing, overrun;

void func(void)
{
	R_PG_Clock_Set();		//The clock-generation circuit has to be set first.

	R_PG_SCI_Set_C0();		//Set up SCI0.

	//Start the data reception
	R_PG_SCI_StartReceiving_C0(data_re, 10);
}

void Sci0ReFunc(void)
{
	//Acquire the reception error flags
	R_PG_SCI_GetReceptionErrorFlag_C0(&parity, &framing, &overrun);

	//Clear the serial reception error flag
	R_PG_SCI_ClearReceptionErrorFlag_C0();
}
```

5.17.23 R_PG_SCI_GetTransmitStatus_C<*channel number*>

Definition	bool R_PG_SCI_GetTransmitStatus_C<*channel number*> (bool * complete) <*channel number*>: 0, 1, 2, 3 and 12
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 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, 1, 2, 3 and 12
RPDL function	R_SCI_GetStatus

Details

- This function acquires the state of transmission.

Example

```
//Include "R_PG_<PDG 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 );
}
```

5.17.24 R_PG_SCI_StopModule_C<*channel number*>

Definition	bool R_PG_SCI_StopModule_C<*channel number*> (void) <*channel number*>: 0, 1, 2, 3 and 12
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, 1, 2, 3 and 12

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_<PDG 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
}
```

5.18 I^{2}C Bus Interface (RIIC)

5.18.1 R_PG_I2C_Set_C<*channel number*>

Definition: bool R_PG_I2C_Set_C<*channel number*> (void)
<*channel number*>: 0, 1

Description: Set up a I^{2}C 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_Set, R_IIC_Create

Details:
- Releases an I^{2}C bus interface channel from the module-stop state, makes initial settings.
- Function R_PG_Clock_Set must be called before any use of this function.

Example: RIIC0 has been set in the GUI.

```
//Include "R_PG_<PDG 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
}
```

5.18.2 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 I^2C bus interface channel in GUI.

Parameter

bool addr_10bit	Slave address format (1: 10bit 0: 7bit)
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 receive.
  Create the notification function as follows:
  void <*name of the notification function*> (void)
  For the notification function, note the contents of this chapter end, 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, [7:1] 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 aquired 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_<PDG project name>.h" to use this function.
#include "R_PG_default.h"

uint8_t iic_data[10];    //The storage area for the received data

void func(void)
{
       R_PG_Clock_Set();    //The clock-generation circuit has to be set first
       R_PG_I2C_Set_C0();    //Set up RIIC0
      R_PG_I2C_MasterReceive_C0(    //Master reception
          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
      );

      R_PG_I2C_StopModule_C0();    //Stop RIIC0
}
```

5.18.3 R_PG_I2C_MasterReceiveLast_C<*channel number*>

Definition

bool R_PG_I2C_MasterReceiveLast_C< *channel number* >
(uint8_t* data)
< *channel number* >: 0

Description

Complete a master reception process

Conditions for output

- The function of master is selected for an I^{2}C bus interface channel in GUI.
- Select DMAC or 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 DMAC] or [Transfer the received serial data by DTC] is selected as a master reception method.
- In the master reception process that has used the DMAC or 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 DMAC or DTC transfer completes, call this function from DMAC or 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 DMAC" is selected as the master reception method in RIIC0 setting.
- DMAC0 is set as follows

  Transfer request source : ICRXI0(receive data full interrupt of TIIC0)
  Transfer system : Single-operand transfer
  Unit data size : 1 byte
  Single operand data count : 1
  Total transfer data size : 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
  DMA interrupt notification fuction name : Dmac0IntFunc

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

void Dmac0IntFunc(){
    uint8_t data; //Strage area of extra data

    //Isse NACK and STOP condition and complete the reception
    R_PG_I2C_MasterReceiveLast_C0( &data );
```

```
}
void func(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 DMAC0
    R_PG_DMAC_Set_C0();

    //Activate the DMAC0
    R_PG_DMAC_Activate_C0();

    //Master reception
    R_PG_I2C_MasterReceive_C0(
        0,      //Slave address format
        6,      //Slave address
        PDL_NO_PTR, // For DMAC transfer, set PDL_NO_PTR
        10      // The number of the data (For DMAC transfer, set 0)
    );
}
```

5.18.4 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 I^{2}C bus interface channel in GUI.

Parameter:

bool addr_10bit	Slave address format (1: 10bit 0: 7bit)
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 this chapter end, 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, [7:1] 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 aquired 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_<PDG 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();
}
```

5.18.5 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 I^2C bus interface channel in GUI.

Parameter

bool addr_10bit	Slave address format (1: 10bit 0: 7bit)
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 this chapter end, 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, [7:1] 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 aquired 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_<PDG 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();
}
```

5.18.6 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 I^2C 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: RIIC0 has been set in the GUI.

```
//Include "R_PG_<PDG 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();
}
```

5.18.7 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 I^2C bus interface channel in GUI.

Parameter:

bool *busy	The address of storage area for the bus busy detection flag

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.

Bus busy detection flag

0	The I^2C bus is released (bus free state)
1	The I^2C bus is occupied (bus busy state or in the bus free state)

Example: RIIC0 has been set in the GUI.

```
//Include "R_PG_<PDG 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
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();

    // 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
    );
```

5.18.8 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 I^2C 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 this chapter end, 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 aquired 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_<PDG 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_I2C_GetTR_C0(&transmit);

    //Get a detected address
    R_PG_I2C_GetDetectedAddress_C0(&addr0, &addr1, 0, 0, 0, 0);

    if (start && transmit && address0) {
        R_PG_I2C_SlaveSend_C(
            iic_data_tr_0,
            10
        );
    }

    else if (start && read && address1) {
        R_PG_I2C_SlaveSend_C(
            iic_data_tr_1,
            10
        );
    }
}
```

5.18.9 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 I^2C 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*>

5.18.10 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 I^2C bus interface channel in GUI.

Parameter

bool *addr0	The address of storage area for slave address 0 detection flag
bool *addr1	The address of storage area for slave address1 detection flag
bool *addr2	The address of storage area for slave address 2 detection flag
bool *general	The address of storage area for general call address detection flag
bool *device	The address of storage area for device-ID command detection flag
bool *host	The address of 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.
- 1 is set to detected address

Example Refer to the example of R_PG_I2C_SlaveMonitor_C<*channel number*>

5.18.11 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 I^2C bus interface channel in GUI.

Parameter

bool * transmit	The address of storage area for the transmit mode 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.
- 1 is set to detected address.
- This function acquires the the transmit/receive mode.

Transmit mode flag

0	Receive mode
1	Transmit mode

Example Refer to the example of R_PG_I2C_SlaveMonitor_C<*channel number*>

5.18.12 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 storage area for a NACK detection flag
bool *stop	The address of storage area for a stop condition detection flag
bool *start	The address of storage area for a start condition detection flag
bool *lost	The address of storage area for an arbitration lost
bool *timeout	The address of 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.
- 1 is set to detected event.

Example
Refer to the example of R_PG_I2C_SlaveMonitor_C<*channel number*>

5.18.13 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 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_<PDG 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 );
}
```

5.18.14 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 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 data written in I^{2}C Bus Transmit Data Register (ICDRT).
- 0 is acquired when the number of transmission specified to the transmitting function is completed.

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_<PDG 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 );
}
```

5.18.15 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_<PDG 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();
	}
}
```

5.18.16 R_PG_I2C_StopModule_C<*channel number*>

Definition bool R_PG_I2C_StopModule_C<*channel number*> (void)

<*channel number*>: 0

Description Shut down the I^2C 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 I^2C 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_<PDG 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();
}
```

5.19 Serial Peripheral Interface (RSPI)

5.19.1 R_PG_RSPI_Set_C<*channel number*>

Definition: bool R_PG_RSPI_Set_C<*channel number*> (void)
<*channel number*>: 0, 1

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, 1

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_<PDG 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
}
```

5.19.2 R_PG_RSPI_SetCommand_C<*channel number*>

Definition: bool R_PG_RSPI_SetCommand_C<*channel number*> (void)
<*channel number*>: 0, 1

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, 1

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*>

5.19.3 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, 1

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, 1

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, 1

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 this chapter end, 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_<PDG 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();
}
```

5.19.4 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, 1

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, 1

The DTC/DMAC transfer is selected for the transfer method

bool R_PG_RSPI_TransferAllData_C<*channel number*>
(uint16_t sequence_loop_count)
<*channel number*>: 0, 1

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, 1

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_<PDG 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();
}
```

5.19.5 R_PG_RSPI_GetStatus_C<*channel number*>

Definition bool R_PG_RSPI_GetStatus_C<*channel number*> (bool * idle)
<*channel number*>: 0, 1

Description Acquire the transfer status

Parameter

bool * idle	The address of storage area for the idle flag (0: Idle state 1: Transfer state)

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output R_PG_RSPI_C<*channel number*>.c
<*channel number*>: 0, 1

RPDL function R_SPI_GetStatus

Details

- Acquires the transfer status.
- 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_<PDG 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 );
	}while( idle );
}
```

5.19.6 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, 1

Description: Acquire the error flags

Parameter

bool * over_run	The address of storage area for the overrun error flag
bool * mode_fault	The address of storage area for the mode fault error flag
bool * parity_error	The address of 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, 1

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*>

5.19.7 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, 1

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 storage area for the current command pointer value (0 to 7)
uint8_t * error_command	The address of 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, 1

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_<PDG 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( 0, &error_command );

                // Processing when an error is detected
        }
}
```

5.19.8 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, 1

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, 1

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_<PDG project name>.h” to use this function.
#include "R_PG_default.h"

void func(void)
{
	R_PG_RSPI_LoopBackDirect_C0();	//Set loopback mode
}
```

5.19.9 R_PG_RSPI_StopModule_C<*channel number*>

Definition bool R_PG_RSPI_StopModule_C<*channel number*> (void)
<*channel number*>: 0, 1

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, 1

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*>.

5.20 CRC Calculator (CRC)

5.20.1 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_<PDG 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_ClearResult();      // Clear the CRC alculation result
	R_PG_CRC_StopModule();    // Shutdown the CRC unit
}
```

5.20.2 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.

5.20.3 R_PG_CRC_GetResult

Definition bool R_PG_CRC_GetResult (uint16_t * data)

Description Get the CRC calculation result

Parameter

uint16_t * data	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 result of calculation

Example Refer to the example of R_PG_CRC_Set.

5.20.4 R_PG_CRC_ClearResult

Definition bool R_PG_CRC_ClearResult (void)

Description Clear the CRC calculation result

Parameter None

Return value

true	Setting was made correctly.
false	Setting failed.

File for output R_PG_CRC.c

RPDL function R_CRC_Read

Details
- This function clears the the result of calculation

Example Refer to the example of R_PG_CRC_Set.

5.20.5 R_PG_CRC_StopModule

Definition bool R_CRC_Destroy (uint16_t * data)

Description Shut down CRC calculator

Parameter None

Return value

true	Acquisition succeeded
false	Acquisition 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.

5.21 12-Bit A/D Converter (S12ADB)

5.21.1 R_PG_ADC_12_Set_S12AD*<unit number>*

Definition: bool R_PG_ADC_12_Set_S12AD<*unit number*> (void)
<*unit number*>:0, 1

Description: Sets 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_S12AD<*unit number*>.c
<*unit number*>:0, 1

RPDL function: R_ADC_12_Set, R_ADC_12_Create

Details:

- Releases the 12-bit A/D converter from the module-stop state, makes initial settings, and places the converter 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_StartConversion_S12AD<*unit number*>.
- Before calling this function, call R_PG_Clock_Set to set the clock.
- 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, the function having the specified name will be called when an interrupt request is conveyed to the CPU. Create the interrupt notification function as follows:
  void <*name of the interrupt notification function*> (void)
  For notes on interrupt notification functions, refer to “Notes on Notification Functions” provided at the end of this section.

Example:

```
// Include "R_PG_<PDG 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_12_Set_S12AD0();      // Set up the 12-bit A/D converter (S12AD0).
}
```

5.21.2 R_PG_ADC_12_StartConversion_S12AD<*unit number*>

Definition: bool R_PG_ADC_12_StartConversion_S12AD<*unit number*> (void)
<*unit number*>:0, 1

Description: Starts A/D conversion

Parameter: None

Return value:

true	Setting was made correctly.
false	Setting failed.

File for output: R_PG_ADC_12_S12AD<*unit number*>.c
<*unit number*>:0, 1

RPDL function: R_ADC_12_Control

Details:
- Starts A/D conversion by an A/D converter.

Example: The following setting has been made through the GUI.
- Select the software trigger as the activation source.

```
// Include "R_PG_<PDG 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_12_Set_S12AD0();    // Set up the 12-bit A/D converter (S12AD0).

    // A software trigger starts A/D conversion.
    R_PG_ADC_12_StartConversion_S12AD0();
}
```

5.21.3 R_PG_ADC_12_StopConversion_S12AD<*unit number*>

Definition: bool R_PG_ADC_12_StopConversion_S12AD<*unit number*> (void)
<*unit number*>:0, 1

Description: Stops A/D conversion

Parameter: None

Return value:

true	Stopping conversion succeeded.
false	Stopping conversion failed.

File for output: R_PG_ADC_12_S12AD<*unit number*>.c
<*unit number*>:0, 1

RPDL function: R_ADC_12_Control

Details:
- Stops A/D conversion in the continuous scan mode. In other modes, 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_S12AD0.

Example: The following setting has been made through the GUI.
- Select the continuous scan mode as the operating mode.

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

void func1(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).
}

void func2(void)
{
    // Stop continuous scanning.
    R_PG_ADC_12_StopConversion_S12AD0();
}
```

5.21.4 R_PG_ADC_12_GetResult_S12AD<*unit number*>

Definition bool R_PG_ADC_12_GetResult_S12AD<*unit number*> (uint16_t * result)
<*unit number*>:0, 1
Double torigger mode
bool R_PG_ADC_12_GetResult_S12AD<*unit number*>
(uint16_t * result, uint16_t * result_dbl_self, uint16_t * result_dbl_a, uint16_t * result_dbl_b)
<*unit number*>:0, 1

Description Gets the result of A/D conversion of an analog input

Parameter

uint16_t * result	Destination for storage of the result of A/D conversion (ADDR)
uint16_t * result_dbl_self	Destination for storage of the result of A/D conversion (ADDBLDR)
uint16_t * result_dbl_a	Destination for storage of the result of A/D conversion (ADDBLDRA)
uint16_t * result_dbl_b	Destination for storage of the result of A/D conversion (ADDBLDRB)

Return value

true	Acquisition of the result succeeded.
false	Acquisition of the result failed.

File for output R_PG_ADC_12_S12AD<*unit number*>.c
<*unit number*>:0, 1

RPDL function R_ADC_12_Read

Details
- At least two 4-byte spaces are needed for storage of the acquired result of A/D conversion of an analog input.
- 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 The following settings have been made through the GUI.
- Analog input channel was selected as conversion target in the GUI.
- S12ad0AIntFunc was specified as A/D conversion end interrupt notification function name in the GUI.

```
// Include "R_PG_<PDG 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_12_Set_S12AD0();     // Set up the 12-bit A/D converter (S12AD0).
}

// A/D-conversion end interrupt notification function
void S12ad0AIntFunc(void)
{
    uint16_t result[4];     // Destination for storing the result of A/D conversion.

    // Acquire the results of A/D conversion.
    R_PG_ADC_12_GetResult_S12AD0( result, 0, 0, 0 );
}
```

5.21.5 R_PG_ADC_12_GetResult_SelfDiag_S12AD<*unit number*>

Definition
bool R_PG_ADC_12_GetResult_SelgDiag_S12AD<*unit number*> (uint16_t * result)
<*unit number*>:0, 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_S12AD<*unit number*>.c
<*unit number*>:0, 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_<PDG 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[4];         // 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_StartConversion_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];
}
```

5.21.6 R_PG_ADC_12_StartComparator_S12AD*<unit number>*

Definition
bool R_PG_ADC_12_StartComparator_S12AD0(bool an000, bool an001, bool an002)
bool R_PG_ADC_12_StartComparator_S12AD1(bool an100, bool an101, bool an102)

Description
Start the comparator

Parameter

bool an000	Start setting for the comparator of AN000. (1:start 0:no change)
bool an001	Start setting for the comparator of AN001 (1:start 0:no change)
bool an002	Start setting for the comparator of AN002 (1:start 0:no change)
bool an100	Start setting for the comparator of AN100. (1:start 0:no change)
bool an101	Start setting for the comparator of AN101 (1:start 0:no change)
bool an102	Start setting for the comparator of AN102 (1:start 0:no change)

Return value

True	Setting was made correctly
False	Setting failed

File for output
R_PG_ADC_12_S12AD*<unit number>*.c
<unit number>:0, 1

RPDL function
R_ADC_12_Control

Details
- Start the comparator of specified analog input.

Example

```
// Include "R_PG_<PDG 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.

        // Set up the comparator
        R_PG_ADC_12_Set_S12AD0();

        // Start setting for the comparator
        R_PG_ADC_12_StartComparator_S12AD( 1, 0, 0 )
}

//Comparater detection interrupt notification function
void S12ad0Cmp0IntFunc(void)
{
        bool an000;                    //strage of the comparator detection flag for AN000

        // Acquire the error flags
        R_PG_ADC_12_ GetComparatorStatusFlag _S12AD0( &an000, 0, 0 );

        // Stop the compararot
        R_PG_ADC_12_StopCompararot_S12AD( 1, 0, 0 );
}
```

5.21.7 R_PG_ADC_12_StopComparator_S12AD<*unit number*>

Definition
bool R_PG_ADC_12_StopComparator_S12AD0(bool an000, bool an001, bool an002)
bool R_PG_ADC_12_StopComparator_S12AD1(bool an100, bool an101, bool an102)

Description
Stop the comparator

Parameter

bool an000	Stop setting for the comparator of AN000. (1:stop 0:no change)
bool an001	Stop setting for the comparator of AN001 (1:stop 0:no change)
bool an002	Stop setting for the comparator of AN002 (1:stop 0:no change)
bool an100	Stop setting for the comparator of AN100. (1:stop 0:no change)
bool an101	Stop setting for the comparator of AN101 (1:stop 0:no change)
bool an102	Stop setting for the comparator of AN102 (1:stop 0:no change)

Return value

true	Setting was made correctly
false	Setting failed

File for output
R_PG_ADC_12_S12AD<*unit number*>.c
<*unit number*>:0, 1

RPDL function
R_ADC_12_Control

Details
- Stop the comparator of specified analog input.

Example
Refer to the example of R_PG_ADC_12_StartComparator_S12AD<*unit number*>.

5.21.8 R_PG_ADC_12_GetComparatorStatusFlag_S12AD<*unit number*>

Definition
bool R_PG_ADC_12_GetComparatorStatusFlag_S12AD0
(bool * an000, bool * an001, bool * an002)
bool R_PG_ADC_12_GetComparatorStatusFlag_S12AD1
(bool * an100, bool * an101, bool * an102)

Description
Acquire the comparater detection flags

Parameter

bool * an000	Destination for storage of the result of comparator detection flag for AN000
bool * an001	Destination for storage of the result of comparator detection flag for AN001
bool * an002	Destination for storage of the result of comparator detection flag for AN002
bool * an100	Destination for storage of the result of comparator detection flag for AN100
bool * an101	Destination for storage of the result of comparator detection flag for AN101
bool * an102	Destination for storage of the result of comparator detection flag for AN102

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output
R_PG_ADC_12_S12AD<*unit number*>.c
<*unit number*>:0, 1

RPDL function
R_ADC_12_Read

Details
- This function acquires the comparator detection flag of Voltage Detection Circuit.
  Specify 0 for a flag that is not required.

Example
Refer to the example of R_PG_ADC_12_StartComparator_S12AD<*unit number*>.

5.21.9 R_PG_ADC_12_StopModule_S12AD<*unit number*>

Definition: bool R_PG_ADC_12_StopModule_S12AD<*unit number*> (void)

<*unit number*>:0, 1

Description: Shuts 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_S12AD<*unit number*>.c

<*unit number*>:0, 1

RPDL function: R_ADC_12_Destroy

Details:

- Stops the 12-bit A/D converter and places it in the module-stop state.

Example:

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

uint16_t result[4];    // 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_12_Set_S12AD0();        // Set up the 12-bit A/D converter (S12AD0).
}

void func2(void)
{
    // Stop continuous scanning.
    R_PG_ADC_12_StopConversion_S12AD0();

    // Acquire the result of A/D conversion.
    R_PG_ADC_12_GetResult_S12AD0( result );

    // Stop the 12-bit A/D converter (S12AD0).
    R_PG_ADC_12_StopModule_S12AD0();
}
```

5.22 10-Bit A/D Converter (AD)

5.22.1 R_PG_ADC_10_Set_AD<*unit number*>

Definition: bool R_PG_ADC_10_Set_AD<*unit number*> (void)
<*unit number*>:0

Description: Sets up the 10-bit A/D converter

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_Set, R_ADC_10_Create

Details:

- Releases the 10-bit A/D converter from the module-stop state, makes initial settings, and places the converter 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_10_StartConversion_AD<*unit number*>.
- Before calling this function, call R_PG_Clock_Set to set the clock.
- 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, the function having the specified name will be called when an interrupt request is conveyed to the CPU. Create the interrupt notification function as follows:
  void <*name of the interrupt notification function*> (void)
  For notes on interrupt notification functions, refer to “Notes on Notification Functions” provided at the end of this section.

Example:

```
// Include "R_PG_<PDG 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 the 10-bit A/D converter (AD0).
}
```

5.22.2 R_PG_ADC_10_StartConversion_AD<*unit number*>

Definition
bool R_PG_ADC_10_StartConversion_AD<*unit number*> (void)
<*unit number*>:0

Description
Starts A/D conversion

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_Control

Details
- Starts A/D conversion by an A/D converter.

Example
The following setting has been made through the GUI.
- Select the software trigger as the activation source.

```
// Include "R_PG_<PDG 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 the 10-bit A/D converter (AD0).

    // A software trigger starts A/D conversion.
    R_PG_ADC_10_StartConversion_AD0();
}
```

5.22.3 R_PG_ADC_10_StopConversion_AD<*unit number*>

Definition: bool R_PG_ADC_10_StopConversion_AD<*unit number*> (void)
<*unit number*>:0

Description: Stops A/D conversion

Parameter: None

Return value:

True	Stopping conversion succeeded.
False	Stopping conversion failed.

File for output: R_PG_ADC_10_AD<*unit number*>.c
<*unit number*>:0

RPDL function: R_ADC_10_Control

Details:

- Stops A/D conversion in the continuous scan mode. In other modes, 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_AD0.

Example: The following setting has been made through the GUI.

- Select the continuous scan mode as the operating mode.

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

void func1(void)
{
    R_PG_Clock_Set();             // The clock-generation circuit has to be set first.
    R_PG_ADC_10_Set_AD0();     // Set up the 10-bit A/D converter (AD0).
}

void func2(void)
{
    // Stop continuous scanning.
    R_PG_ADC_10_StopConversion_AD0();
}
```

5.22.4 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
Gets the result of A/D conversion of an analog input

Parameter

uint16_t * result	Destination for storage of the result of A/D conversion (ADDR)

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

- At least two 20-byte spaces are needed for storage of the acquired result of A/D conversion of an analog input.
- 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
The following settings have been made through the GUI.

- Analog input channel was selected as conversion target in the GUI.
- Ad0IntFunc was specified as A/D conversion end interrupt notification function name in the GUI.

```
// Include "R_PG_<PDG 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 the 10-bit A/D converter (AD0).
}

// A/D-conversion end interrupt notification function
void Ad0IntFunc(void)
{
    uint16_t result[20];     // Destination for storing the result of A/D conversion.

    // Acquire the results of A/D conversion.
    R_PG_ADC_10_GetResult_AD0( result );
}
```

5.22.5 R_PG_ADC_10_GetResult_SelfDiag_AD<*unit number*>

Definition bool R_PG_ADC_10_GetResult_SelgDiag_AD<*unit number*> (uint16_t * result)
<*unit number*>:0

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_10_AD<*unit number*>.c
<*unit number*>:0

RPDL function R_ADC_10_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 AN0 and AN10 as analog input pins.
- Select the software trigger as the activation source.
- Select right-alignment for data placement.
- Enable the self-diagnosis facility.
- Specify Ad0IntFunc as the A/D-conversion end interrupt notification function.

```
// Include "R_PG_<PDG 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 10-bit A/D conversion
uint16_t adrd_diagst;       // Destination for storing the self-diagnosis status information
uint16_t result[20];        // Destination for storing the result of A/D conversion on AN0
                            // and AN10
uint16_t result_an0;        // Destination for storing the result of A/D conversion on AN0
uint16_t result_an10;       // Destination for storing the result of A/D conversion on AN10

void func(void)
{
    R_PG_Clock_Set();               // The clock-generation circuit has to be set first.
    R_PG_ADC_10_Set_AD0();   // Set up the 12-bit A/D converter (S12AD0).

    // A software trigger starts A/D conversion.
    R_PG_ADC_10_StartConversion_AD0();
}

// A/D-conversion end interrupt notification function
void Ad0IntFunc(void)
{
    // Acquire the results of A/D conversion as part of self diagnosis.
    R_PG_ADC_10_GetResult_SelfDiag_AD0( &result_selfdiag );

    adrd_ad = (result_selfdiag & 0x0fff);
    adrd_diagst = (result_selfdiag >> 14);

    // Acquire the result of A/D conversion on AN0 and AN10.
    R_PG_ADC_10_GetResult_AD0( result );

    result_an0 = result[0];
    result_an10 = result[10];
}
```

5.22.6 R_PG_ADC_10_StopModule_AD<*unit number*>

Definition: bool R_PG_ADC_10_StopModule_AD<*unit number*> (void)
<*unit number*>:0

Description: Shuts down the 10-bit A/D converter

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 the 10-bit A/D converter and places it in the module-stop state.

Example:

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

uint16_t result[20];   // 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 the 10-bit A/D converter (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( result );

    // Stop the 10-bit A/D converter (AD0).
    R_PG_ADC_10_StopModule_AD0();
}
```

5.23 D/A Converter (DAa)

5.23.1 R_PG_DAC_Set_C<*channel number*>

Definition: bool R_PG_DAC_Set_S<*channel number*> (void)
<*channel number*>:0, 1

Description: Sets up the D/A converter channel

Parameter: None

Return value:

True	Setting was made correctly.
False	Setting failed.

File for output: R_PG_DAC_C<*channel number*>.c
<*channel number*>:0, 1

RPDL function: R_DAC_10_Create

Details:

- Set up a D/A converter channel.
- Release the D/A converter from the module-stop state.
- The conversion result of data register’s initial value (=0) after the module-stop state is released is output from the analog output pin.
- If the output begins after specifying an initial value, use R_DAC_SetWithInitialValue_C<*channel number*>.
- If [D/A converter operation synchronizes with 10-bit A/D converter operation] is selected in GUI, do not call this function when a 10-bit A/D conversion is in progress.

Example:

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

void func1(void)
{
    //Set up DA0 pin.
    R_PG_DAC_Set_C0();
}
void func2( uint16_t output_val )
{
    //Change D/A conversion value
    R_PG_DAC_ControlOutput_C0( output_val );
}
```

5.23.2 R_PG_DAC_SetWithInitialValue_C<*channel number*>

Definition: bool R_PG_DAC_SetWithInitialValue_C<*channel number*> (uint16_t data)
<*channel number*>:0, 1

Description: An initial value of the data register is specified, and D/A converter channel iis set.

Parameter:

uint16_t data	An initial value of the data register

Return value:

true	Setting was made correctly.
false	Setting failed.

File for output: R_PG_DAC_C<*channel number*>.c
<*channel number*>:0, 1

RPDL function: R_DAC_10_Create

Details:

- The D/A converter channel is set specifying an initial value of the data register, and the output begins.
- Releases the D/A converter from the module-stop state.
- If [D/A converter operation synchronizes with 10-bit A/D converter operation] is selected in GUI, do not call this function when a 10-bit A/D conversion is in progress.

Example:

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

void func1( uint16_t initial__val )
{
    //Setting up DA0 pin, and output begins.
    R_PG_DAC_SetWithInitialValue_C0( initial_val );
}

void func2( uint16_t output_val )
{
    //Change D/A conversion value.
    R_PG_DAC_ControlOutput_C0( output_val );
}
```

5.23.3 R_PG_DAC_ControlOutput_C<*channel number*>

Definition
bool R_PG_DAC_ControlOutput_C<*Channel number*> (uint16_t data)
<*unit number*>:0

Description
Stops A/D conversion

Parameter

uint16_t data	The value to be written to the data register.

Return value

True	Setting was made correctly.
False	Setting failed.

File for output
R_PG_DAC_C<*channel number*>.c
<*channel number*>:0, 1

RPDL function
R_DAC_10_Write

Details
- Writes the D/A conversion value to the data register.

Example
Refer to the example of R_PG_DAC_Set_C<*channel number*>

5.23.4 R_PG_DAC_StopOutput_C<*channel number*>

Definition: bool R_PG_DAC_StopOutput_C<*channel number*> (void)
<*channel number*>:0, 1

Description: Stop the analog signal output.

Parameter: None

Return value:

true	Acquisition of the result succeeded.
false	Acquisition of the result failed.

File for output: R_PG_DAC_C<*channel number*>.c
<*channel number*>:0, 1

RPDL function: R_DAC_10_Destroy

Details:
- Disable D/A converter channel.
- Once both channels are disabled, the module is put into the power-down state.

Example:

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

void func1( uint16_t initial_val )
{
    //Setting up DA0 pin, and output begins.
    R_PG_DAC_SetWithInitialValue_C0( initial_val );
}

void func2(void)
{
    //Stop the analog signal output.
    R_PG_DAC_StopOutput_C0();
}
```

5.24 Data Operation Circuit (DOC)

5.24.1 R_PG_DOC_Set

Definition bool R_PG_DOC_Set(void)

Description Set up the Data Operation Cicrcuit

Parameter None

Return value

true	Setting was made correctly.
false	Setting failed.

File for output R_PG_DOC.c

RPDL function R_DOC_Create

Details
- Releases the DOC from the module-stop and makes initial settings.
- In Addition Mode an interrupt is generated if the result of the addition exceeds FFFFh.
- In Subtraction Mode an interrupt is generated if the result of the subtraction is less than zero.
- In Comparison Mode an interrupt is generated when the comparison criteria (Match or Mismatch) is met.
- After calling the interrupt notification function the DOC flag is automatically cleared.

Example The following settings have been made through the GUI.
- [Data comparison mode] is selected for the operating mode
- [Detects match as a result of data comparison] is selected for the detection condition
- Comparison reference is 1
- DopcfIntFunc was specified as the interrupt notification function name

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

uint16_t input_data[10]={1,0,0,1,0,0,0,0,0,1};
uint16_t comp_match_cnt=0;

void func(void)
{
    R_PG_DOC_Set();    //Set up the data operation circuit
    R_PG_DOC_InputData(input_data, 10);    //Input data
}

//Data Operation Circuit interrupt notification function
void DopcfIntFunc(void)
{
    comp_match_cnt++;
}
```

5.24.1 R_PG_DOC_GetStatusFlag

Definition bool R_PG_DOC_GetStatusFlag (bool * status)

Description Acquire the status flag of data operation

Parameter

bool * statust	The address of the storage area for the operation status flag

Return value

true	Acquisition of the status flag succeeded.
false	Acquisition of the status flag failed.

File for output R_PG_DOC.c

RPDL function R_DOC_Read

Details

- Acquires the flag of operation result status.
-

Example Refer to the example of R_PG_DOC_StopModule

5.24.2 R_PG_DOC_GetResult

Definition bool R_PG_DOC_GetResult (uint16_t * result)

Description Acquire the result of data operation

Parameter

uint16_t * result	The address of the storage area for the operation result

Return value

true	Acquisition of the result succeeded.
false	Acquisition of the result failed.

File for output R_PG_DOC.c

RPDL function R_DOC_Read

Details

- Acquires the value of DODSR (DOC Data Setting Register).
- The content of the acquired value of each operating mode is different as follows:
  Data comparison mode :Comparison reference
  Data addition mode :The result of data addition
  Data subtraction mode :The result of data subtraction

Example Refer to the example of R_PG_DOC_StopModule

5.24.3 R_PG_DOC_InputData

Definition bool R_PG_DOC_InputData (uint16_t * data, uint16_t count)

Description Input data

Parameter

uint16_t * data	The address of the storage area for the input data
uint16_t count	The number of the input data

Return value

true	Setting was made correctly.
false	Setting failed.

File for output R_PG_DOC.c

RPDL function R_DOC_Write

Details
- Data for the operation is set to DODIR (DOC Data Input Register).

  Data comparison mode :The compared data is set
  Data addition mode :The added data is set
  Data subtraction mode :The subtracted data is set

Example Refer to the example of R_PG_DOC_Set

5.24.4 R_PG_DOC_UpdateData

Definition bool R_PG_DOC_UpdateData (uint16_t data)

Description Update data

Parameter

uint16_t data	Data for update

Return value

true	Setting was made correctly.
false	Setting failed.

File for output R_PG_DOC.c

RPDL function R_DOC_Control

Details

- Data for the operation is set to DODIR (DOC Data Input Register).

  Data comparison mode :The compared data is set
  Data addition mode :The added data is set
  Data subtraction mode :The subtracted data is set

Example The following settings have been made through the GUI.

- [Data comparison mode] is selected for the operating mode
  [Detects match as a result of data comparison] is selected for the detection condition
  Comparison reference is 0
  DopcfIntFunc was specified as the interrupt notification function name

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

uint16_t input_data[10]={1,0,0,1,0,0,0,0,0,1};
uint16_t comp_match_cnt=0;
uint16_t comp_match_0, comp_match_1;

void func(void)
{
	R_PG_DOC_Set();    //Set up the data operation circuit
	R_PG_DOC_InputData(input_data, 10);    //Input data

	comp_match_0 = comp_match_cnt;

	R_PG_DOC_UpdateData(1);    //Update data
	R_PG_DOC_InputData(input_data, 10);    //Input data

	comp_match_1 = comp_match_cnt - comp_match_0;
}

//Data Operation Circuit interrupt notification function
void DopcfIntFunc(void)
{
	comp_match_cnt++;
}
```

5.24.5 R_PG_DOC_StopModule

Definition bool R_PG_DOC_StopModule (void)

Description Disable the data operation circuit

Parameter None

Return value

true	Setting was made correctly.
false	Setting failed.

File for output R_PG_DOC.c

RPDL function R_DOC_Destroy

Details
- Enable the DOC module stop state.

Example The following settings have been made through the GUI.
- [Data addition mode] is selected for the operating mode
- Initial value of addition or subtraction result is 0

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

uint16_t result;
uint16_t data=0x0000;

void func(void)
{
    bool status;

    //Set up the data operation circuit
    R_PG_DOC_Set();

    while(1){
        //Input data
        R_PG_DOC_InputData(&data, 1);

        //Acquire the status of the data operation circuit
        R_PG_DOC_GetStatusFlag(&status);

        if(status == true){
                break;
        }

        //Acquire the result of data operation
        R_PG_DOC_GetResult(&result);

        data++;
    }

    //Disable the data operation circuit
    R_PG_DOC_StopModule();
}
```

5.25 Notes on Notification Functions

5.25.1 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 RPDL 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:

1. 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.
2. 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.

5.25.2 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 RPDL.
If DSP instructions are being utilised in the users' code, notification functions which are called by the interrupt handlers in RPDL should either

1. Avoid using instructions which modify the ACC register.
2. Take a copy of the ACC register and restore it before exiting the callback function.

6. Registering Files with the IDE(HEW, CubeSuite+ or e2 studio) and Building Them

Note the following points when registering the files generated by the PDG with the IDE and building them.

(1) Source files generated by the PDG 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 PDG 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 PDG registers with the IDE, is overwritten when the PDG generates source files.

(4) The RPDL 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 RPDL 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 PDG-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.

RX63T 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

Colophon 3.0

RX63T Group
Peripheral Driver Generator
Reference Manual

Renesas Electronics Corporation

R20UT2274EJ0102