User's Manual

RX62N Group

Peripheral Driver Generator
Reference Manual

Renesas Electronics
www.renesas.com

Rev.1.04 May 2014

Notice

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

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

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

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

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

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

Introduction

This manual was written to explain how to make the peripheral I/O drivers on the Peripheral Driver Generator for RX62N. 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 RX62N group, peripheral modules and endian.

(1) Products

Part No.	Package	Part No.	Package
R5F562N8BDBG	PLBG0176GA-A	R5F562N8ADFB	PLQP0144KA-A
R5F562N8BDLE	PTLG0145JB-A	R5F562N8ADFP	PLQP0100KB-A
R5F562N8BDFB	PLQP0144KA-A	R5F562N7ADBG	PLBG0176GA-A
R5F562N8BDFP	PLQP0100KB-A	R5F562N7ADLE	PTLG0145JB-A
R5F562N7BDBG	PLBG0176GA-A	R5F562N7ADFB	PLQP0144KA-A
R5F562N7BDLE	PTLG0145JB-A	R5F562N7ADFP	PLQP0100KB-A
R5F562N7BDFB	PLQP0144KA-A	R5F56108VNFPP	LQP0144KAA
R5F562N7BDFP	PLQP0100KB-A	R5F56107VNFPP	LQP0144KAA
R5F562N8ADBG	PLBG0176GA-A	R5F56106VNFPP	LQP0144KAA
R5F562N8ADLE	PTLG0145JB-A	R5F56104VNFPP	LQP0144KAA

(2) Peripheral Modules

Voltage Detection Circuit (LVD)	8-Bit Timer (TMR)
Clock Generation Circuit	Compare Match Timer (CMT)
Low Power Consumption	Realtime Clock (RTC)
Interrupt Control Unit (ICUa), Exceptions	Watchdog Timer (WDT)
Buses	Independent Watchdog Timer (IWDT)
DMA Controller (DMACA)	Serial Communications Interface (SCIa)
EXDMA Controller (EXDMAC)	CRC Calculator (CRC)
Data Transfer Controller (DTCa)	I2C Bus Interface (RIIC)
I/O Ports	Serial Peripheral Interface (RSPI)
Multi-Function Timer Pulse Unit 2 (MTU2) *	12-Bit A/D Converter (S12AD)
Port Output Enable 2 (POE2)	10-Bit A/D Converter (ADa)
Programmable Pulse Generator (PPG)	D/A Converter

* Complementary PWM mode and reset-synchronized PWM are not supported

(3) Endian

Little endian / Big endian

1.2 Tool requirements

The following tools are required for this version of RX62N group Peripheral Driver Generator.

- RX Family C/C++ Compiler Package V.1.02 Release 01
- RX62N Group Renesas Peripheral Driver Library V.1.10 (Bundled in Peripheral Driver Generator)

2. Creating a new project

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

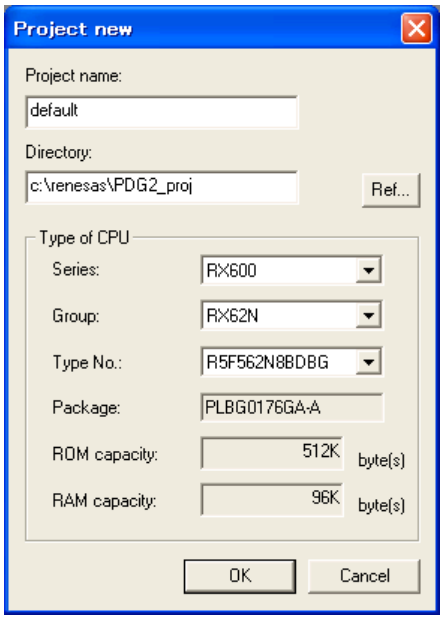


Fig 2.1 New project dialog box

For RX62N group, select [RX600] as a series and select [RX62N] as a group. The package type, ROM capacity and RAM capacity of selected product are displayed.

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

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

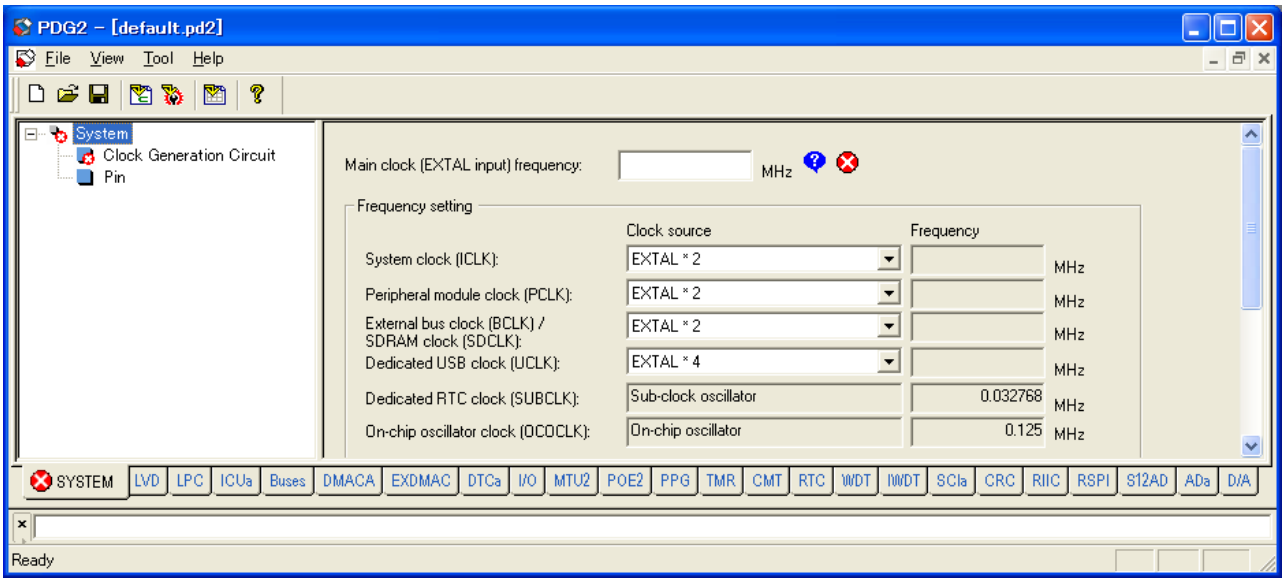


Fig 2.2 Error display of new project

Set the frequency of the lock to be used here.

3. Setting Up the Peripheral Modules

3.1 Main Window

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

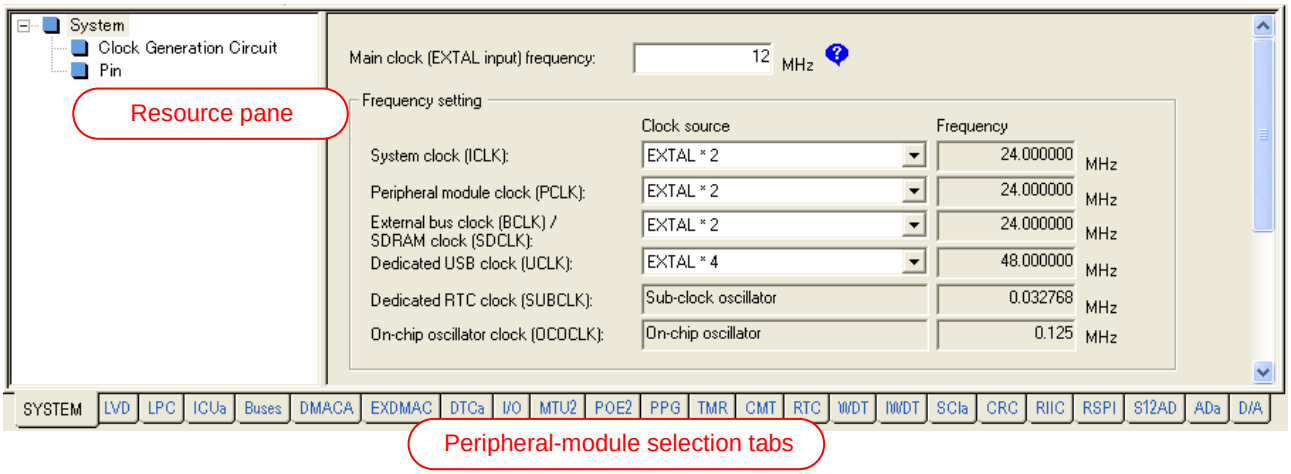


Figure 3.1 Display in the Main Window (Example)

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

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

Tab	Resource pane	Corresponding Peripheral Module or Function
SYSTEM	Clock Generation Circuit	Clock Generation Circuit
	Pin	Pinfunctions
LVD	LVD	LVD1 and 2
LPC	Low Power Consumption	Low Power Consumption Functions
ICUa	Interrupts	Interrupt Control Unit (ICUa) (Fastinterrupt, Software Interrupt, External Interrupt (NMI, IRQ0 to IRQ15))
	Exceptions	Exceptions
Buses	CS0 to CS7	CS area (CS0 to CS7)
	SDCS	SDRAM area
	Common settings	Bus Error Monitoring (illegal address access detection and timeout detection)
DMACA	DMACA0 to DMACA3	DMA Controller (DMACA) Channel 0 to 3
EXDMAC	EXDMAC0 and 1	EXDMA Controller (EXDMAC) Channel 0 and 1
DTCa	DTCa	Data Transfer Controller (DTCa)
I/O	Port 0 to Port G	I/O Port 0 to G
MTU2	Unit0 (MTU0 to MTU5)	Multi-Function Timer Pulse Unit 2 (MTU2) Unit 0 (Channel 0 to 5)
*	Unit1 (MTU6 to MTU11)	Multi-Function Timer Pulse Unit 2 (MTU2) Unit 1 (Channel 6 to 11)
POE2	POE2	Port Output Enable 2 (POE2)
PPG	Group 0 to Group 7	Programmable Pulse Generator (PPG) Group 0 to 7
TMR	Unit0 (TMR0 and TMR1)	8-Bit Timer (TMR) Unit 0 (Channel 0 and 1)
	Unit1 (TMR2 and TMR3)	8-Bit Timer (TMR) Unit 1 (Channel 2 and 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)

RTC	RTC	Real time Clock
WDT	WDT	Watchdog Timer
IWDT	IWDT	Independent Watchdog Timer
SCIa	SCI0, 1, 2, 3, 5, 6	Serial Communications Interface (SCIa) Channel 0 to 3 and 5 to 6
CRC	CRC	CRC Calculator (CRC)
RIIC	RIIC0 and RIIC1	I2C Bus Interface (RIIC) Channel 0 and 1
RSPI	RSPI0 and RSPI1	Serial Peripheral Interface (RSPI) Channel 0 and 1
S12AD	S12AD0	12-Bit A/D Converter (S12AD) Unit 0
ADa	ADa0 to ADa3	10-Bit A/D Converter (ADa) Unit 0 to 3
D/A	DA0 and DA1	D/A Converter Channel 0 and 1

* Complementary PWM mode and reset-synchronized PWM are not supported

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

3.2 Pin Functions

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

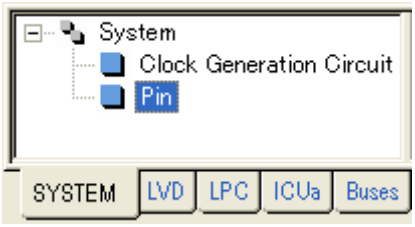


Figure 3.2 Opening the Pin-Function Pane

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

3.2.1 [Pin function] Sheet

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

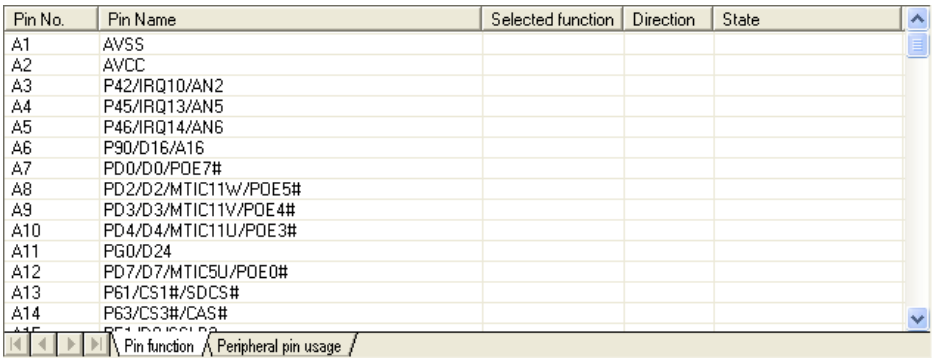


Figure 3.3 Pin-Function Pane ([Pin function] Sheet)

The contents of each column are shown in table 3.2.

Table 3.2 Columns on the [Pin function] Sheet

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

When a peripheral module associated with input to or output from pins has been set up, the current setting is shown on the [Pin function] sheet. In 176-pin LFBGA package, if you have set A/D converter AD0 in the detailed settings pane up so that the input on analog input pin AN0 will be converted, for example, the [Pin function] sheet shows the setting of pin C5 (P40/IRQ8/AN0) as follows.

Pin No.	Pin Name	Selected function	Direction	State
C5	P40/IRQ8/AN0	AN0	Input	

Figre 3.4 Display of selected pin function

Setting up I/O port P40 in this state will cause a conflict between P40 and AN0 and a warning message will be output as shown in figure 3.5.

Pin No.	Pin Name	Selected function	Direction	State
C5	P40/IRQ8/AN0	AN0/P40		Conflicting between different functions

Figre 3.5 Conflict between Pin Functions

Notes

- Pin-by-pin designation of pin functions for RX62N-group MCUs is not possible because the settings of peripheral modules automatically determine the pin functions. The assigned pin functions also cannot be changed in this pane.
- The allocations of some pin functions, however, can be changed on the [Peripheral pin usage] sheet.
- If two or more output functions are enabled on a single pin, the pin only outputs the signal of the function with the highest priority. For details, refer to the hardware manual.

3.2.2 [Peripheral pin usage] Sheet

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

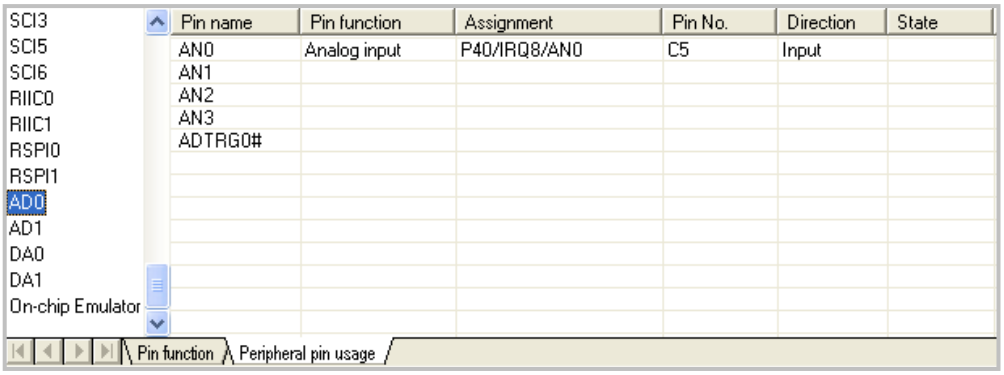


Figure 3.6 Pin-Function Pane ([Peripheral pin usage] Sheet)

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

Table 3.3 Columns on the [Peripheral pin usage] Sheet

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

When a peripheral module associated with input to or output from pins has been set up, the current setting is shown on the [Peripheral pin usage] sheet. In 176-pin LFBGA package, if you have set external interrupt IRQ9 in the detailed settings pane up, for example, the [Peripheral pin usage] sheet shows the setting of pin IRQ9 as follows.

Pin Name	Pin function	Assignment	Pin No.	Direction	State
IRQ9	External interrupt	P01/TMCI0/RxD6/IRQ9	D2	Input	

Figre 3.7 Display of a Pin Function (Example)

Then setting up I/O port P01, which uses the same pin as IRQ9, will cause a conflict between P01 and IRQ9 and a warning message will be output as shown in figure 3.8.

Pin Name	Pin function	Assignment	Pin No.	Direction	State
IRQ9	External interrupt	P01/TMCI0/RxD6/IRQ9	D2	Input	Conflicting with another pin function

Figure 3.8 Confliction between pin functions

Other pins to which IRQ9 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.

Figure 3.9 Drop-Down Button

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

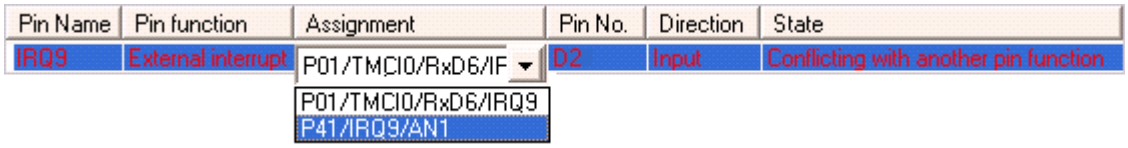


Figure 3.10 Changing the Allocation of a Pin Function

If IRQ9 is assigned to P41/IRQ9/AN1 and that pin is not being used by any other peripheral module, the conflict between P01 and IRQ9 can be resolved.

Pin Name	Pin function	Assignment	Pin No.	Direction	State
IRQ9	External interrupt	P41/IRQ9/AN1	D4	Input	

Figure 3.11 Display after Changing the Allocation

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

4. Specification of Generated Functions

Table 4.1 shows generated functions for the RX62N.

Table 4.1 Generated Functions for the RX62N

Clock-generation circuit

Generated Function	Description
R_PG_Clock_Set	Set up the clocks
R_PG_Clock_Start_SUB	Start the sub-clock oscillator
R_PG_Clock_Stop_SUB	Stop the sub-clock oscillator
R_PG_Clock_GetMainClockStatus	Get the main clock status

Voltage Detection Circuit (LVD)

Generated Function	Description
R_PG_LVD_Set	Set up the voltage detection circuit
R_PG_LVD_GetLVDDetectionFlag	R_PG_LVD_GetLVDDetectionFlag

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

Interrupt controller (ICU)

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

Buses

Generated Function	Description
R_PG_ExtBus_SetBus	Set up the bus pins and 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<*area number*>	Set up CS area
R_PG_ExtBus_DisableArea_CS<*area number*>	Disable CS area
R_PG_ExtBus_SetArea_SDCS	Set up SDRAM area (SDCS)
R_PG_ExtBus_Initialize_SDCS	Start the SDRAM initialization sequence
R_PG_ExtBus_AutoRefreshEnable_SDCS	Enable the SDRAM auto refresh
R_PG_ExtBus_AutoRefreshDisable_SDCS	Disable the SDRAM auto refresh
R_PG_ExtBus_SelfRefreshEnable_SDCS	Enable the SDRAM self refresh
R_PG_ExtBus_SelfRefreshDisable_SDCS	Disable the SDRAM self refresh
R_PG_ExtBus_AccessEnable_SDCS	Enable SDRAM operation
R_PG_ExtBus_AccessDisable_SDCS	Disable SDRAM operation
R_PG_ExtBus_GetStatus_SDCS	Get the status of SDRAM

DMAC controller (DMAC)

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 data transfer (Software trigger)
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	Shut down the all channels of DMAC

EXDMAC controller (EXDMAC)

Generated Function	Description
R_PG_EXDMAC_Set_C<*channel number*>	Set up an EXDMAC channel
R_PG_EXDMAC_Activate_C<*channel number*>	Make the EXDMAC be ready for the start trigger
R_PG_EXDMAC_StartTransfer_C<*channel number*>	Start the data transfer (Software trigger)

R_PG_EXDMAC_Suspend_C<*channel number*>	Suspend the data transfer
R_PG_EXDMAC_GetTransferCount_C<*channel number*>	Get the transfer counter value
R_PG_EXDMAC_SetTransferCount_C<*channel number*>	Set the transfer counter
R_PG_EXDMAC_GetRepeatBlockSizeCount_C<*channel number*>	Get the repeat/block/cluster size counter value
R_PG_EXDMAC_SetRepeatBlockSizeCount_C<*channel number*>	Set the repeat/block/cluster size counter value
R_PG_EXDMAC_ClearInterruptFlag_C<*channel number*>	Get and clear the interrupt request flag
R_PG_EXDMAC_GetTransferEndFlag_C<*channel number*>	Get the transfer end flag
R_PG_EXDMAC_ClearTransferEndFlag_C<*channel number*>	Clear the transfer end flag
R_PG_EXDMAC_GetTransferEscapeEndFlag_C<*channel number*>	Get the transfer escape end flag
R_PG_EXDMAC_ClearTransferEscapeEndFlag_C<*channel number*>	Clear the transfer escape end flag
R_PG_EXDMAC_SetSrcAddress_C<*channel number*>	Set the source address
R_PG_EXDMAC_SetDestAddress_C<*channel number*>	Set the destiation address
R_PG_EXDMAC_SetAddressOffset_C<*channel number*>	Set the address offset
R_PG_EXDMAC_SetExtendedRepeatSrc_C<*channel number*>	Set the source address extended repeat value
R_PG_EXDMAC_SetExtendedRepeatDest_C<*channel number*>	Set the destination address extended repeat value
R_PG_EXDMAC_StopModule_C<*channel number*>	Stop the EXDMAC channel

Data Transfer Controller (DTCa)

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

I/O port

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

Multi-Function Timer Pulse Unit 2 (MTU2)

Generated Function	Description
R_PG_Timer_Set_MTU_U<*unit number*>_C<*channel number*>	Set up the MTU
R_PG_Timer_StartCount_MTU_U<*unit number*>_C<*channel number*>(_<*phase*>)	Start the MTU count operation
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*>	Set the counter value
R_PG_Timer_GetRequestFlag_MTU_U<*unit number*>_C<*channel number*>	Acquire the interrupt request flag
R_PG_Timer_StopModule_MTU_U<*unit number*>	Shut down the MTU unit
R_PG_Timer_GetTGR_MTU_U<*unit number*>_C<*channel number*>	Acquire the general register values
R_PG_Timer_SetTGR_<*general register*>_MTU_U<*unit number*>_C<*channel number*>	Set the general register A value
R_PG_Timer_SetBuffer_AD_MTU_U<*unit number*>_C<*channel number*>	Set A/D converter start request cycle set buffer registers

Port Output Enable 2 (POE2)

Generated Function	Description
R_PG_POE_Set	Set up the POE
R_PG_POE_SetHiZ_MTU<*MTU channel number*>	Place the MTU output pins in high-impedance state
R_PG_POE_GetRequestFlagHiZ_MTU<*MTU channel number*>	Acquire the high-impedance request flags
R_PG_POE_GetShortFlag_MTU<*MTU channel number1*>_<*MTU channel number2*>	Acquire the MTU output short flags
R_PG_POE_ClearFlag_MTU<*MTU channel number*>	Clear the high-impedance request flags and the output short flags

Programmable Pulse Generator (PPG)

Generated Function	Description
R_PG_PPG_StartOutput_U<*unit number*>_G<*group number*>	Set up the PPG and start outputting
R_PG_PPG_StopOutput_U<*unit number*>_G<*group number*>	Stop outputting
R_PG_PPG_SetOutputValue_ U<*unit number*>_G<*group number*>	Set the output value of single group
R_PG_PPG_SetOutputValue_ U<*unit number*>_G<*group number1*>_G<*group number2*>	Set the output value for a pair of groups

8-bit timer (TMR)

Generated Function	Description
R_PG_Timer_Start_TMR_U<*unit number*>(_C<*channel number*>)	Set a TMR and start it counting
R_PG_Timer_HaltCount_TMR_U<*unit number*>(_C<*channel number*>)	Halt counting by a TMR
R_PG_Timer_ResumeCount_TMR_U<*unit number*>(_C<*channel number*>)	Resume counting by a TMR
R_PG_Timer_GetCounterValue_TMR_U<*unit number*>(_C<*channel number*>)	Get the counter value of a TMR
R_PG_Timer_SetCounterValue_TMR_U<*unit number*>(_C<*channel number*>)	Set the counter value of a TMR
R_PG_Timer_GetRequestFlag_TMR_U<*unit number*>(_C<*channel number*>)	Acquire and clear the TMR interrupt flags
R_PG_Timer_StopModule _TMR_U<*unit number*>	Shut down a TMR unit

Compare Match Timer (CMT)

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

R_PG_Timer_SetCounterValue_CMT_U<*unit number*>_C<*channel number*>	Set the CMT counter value
R_PG_Timer_StopModule _CMT_U<*unit number*>	Shut down the CMT unit

Realtime Clock (RTC)

Generated Function	Description
R_PG_RTC_Set	Set up the RTC and start the count operation
R_PG_RTC_Stop	Halt the RTC count operation
R_PG_RTC_Restart	Resume the RTC count operation
R_PG_RTC_SetCurrentTime	Set the current time
R_PG_RTC_SetAlarmTime	Set the alarm time
R_PG_RTC_Alarm_Control	Enable/Disable the alarm
R_PG_RTC_Adjust30sec	Start the 30-second adjustment process
R_PG_RTC_SetPeriodicInterrupt	Set the interval for the periodic interrupt
R_PG_RTC_GetStatus	Get the status of RTC
R_PG_RTC_ClockOut_Disable	Enable the clock output
R_PG_RTC_ClockOut_Enable	Disable the clock output

Watchdog Timer (WDT)

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

Independent Watchdog Timer (IWDT)

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

Serial Communications Interface (SCIa)

Generated Function	Description
R_PG_SCI_Set_C<*channel number*>	Set up a SCIa channel
R_PG_SCI_StartSending_C<*channel number*>	Start the data transmission
R_PG_SCI_SendAllData_C<*channel number*>	Transmit all data
R_PG_SCI_GetSentDataCount_C<*channel number*>	Acquire the number of transmitted data
R_PG_SCI_StartReceiving_C<*channel number*>	Start the data reception
R_PG_SCI_ReceiveAllData_C<*channel number*>	Receive all data
R_PG_SCI_StopCommunication_C<*channel number*>	Stop transmission and reception
R_PG_SCI_GetReceivedDataCount_C<*channel number*>	Acquire the number of received data
R_PG_SCI_GetReceptionErrorFlag_C<*channel number*>	Get the serial reception error flag
R_PG_SCI_GetTransmitStatus_C<*channel number*>	Get the state of transmission
R_PG_SCI_SendTargetStationID_C<*channel number*>	Transmits the ID code of the receiving station
R_PG_SCI_ReceiveStationID_C<*channel number*>	Receives the ID code matches the ID of the

	receiving station itself
R_PG_SCI_StopModule_C<*channel number*>	Shut down a SCIa 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 the result of calculation
R_PG_CRC_StopModule	Shut down CRC Calculator

I2C Bus Interface (RIIC)

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

Serial Peripheral Interface (RSPI)

Generated Function	Description
R_PG_RSPI_Set_C<*channel number*>	Set up a RSPI channel
R_PG_RSPI_SetCommand_C<*channel number*>	Set commands
R_PG_RSPI_StartTransfer_C<*channel number*>	Start the data transfer
R_PG_RSPI_TransferAllData_C<*channel number*>	Transfer all data
R_PG_RSPI_GetStatus_C<*channel number*>	Acquire the transfer status
R_PG_RSPI_GetError_C<*channel number*>	Acquire the error flags
R_PG_RSPI_GetCommandStatus_C<*channel number*>	Acquire the command status
R_PG_RSPI_StopModule_C<*channel number*>	Shut down a RSPI channel
R_PG_RSPI_LoopBack<*loopback mode*>_C<*channel number*>	Set loopback mode

12-Bit A/D Converter (S12AD)

Generated Function	Description
R_PG_ADC_12_Set_S12AD0	Set up the 12-Bit A/D Converter
R_PG_ADC_12_StartConversionSW_S12AD0(void)	Start A/D conversion (Software trigger)
R_PG_ADC_12_StopConversion_S12AD0	Stop A/D conversion
R_PG_ADC_12_GetResult_S12AD0	Acquire the result of A/D conversion

R_PG_ADC_12_StopModule_S12AD0	Shut down the 12-Bit A/D converter

10-Bit A/D Converter (ADa)

Generated Function	Description
R_PG_ADC_10_Set_AD<*unit number*>	Set up the 10-Bit A/D Converter (ADa)
R_PG_ADC_10_StartConversionSW_AD<*unit number*>	Start A/D conversion (software trigger)
R_PG_ADC_10_StopConversion_AD<*unit number*>	Stop A/D conversion
R_PG_ADC_10_GetResult_AD<*unit number*>	Get the result of A/D conversion
R_PG_ADC_10_SetSelfDiag_VREF_<*voltage*>_AD<*unit number*>	Set up the A/D self-diagnostic function
R_PG_ADC_10_StartSelfDiag_AD<*unit number*>	Start the A/D conversion (Self-diagnostic function)
R_PG_ADC_10_StopModule_AD<*unit number*>	Shut down the 10-Bit A/D Converter (ADa)

D/A Converter

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_StopOutput_C<*channel number*>	Stop output
R_PG_DAC_ControlOutput_C<*channel number*>	Input the data
R_PG_DAC_Set_C0_C1	Set up the D/A converter channel (DA0 and DA1)

4.1 Clock-Generation Circuit

4.1.1 R_PG_Clock_Set

Definition: bool R_PG_Clock_Set(void)

Description: Set up the clocks

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Clock.c

RPDL function: R_CGC_Set

Details:

- Sets registers in the clock-generation circuit and multiplication ratios to derive the system clock (ICLK), peripheral module clock (PCLK), external bus clock (BCLK), and SDRAM clock (SDCLK) from EXTAL.
- Sets the oscillation stop detection function.
- Sets the pin output of the external bus clock (BCLK) and the SDRAM clock (SDCLK).
- The sub-clock oscillator shall be stopped in this function. To start or stop the sub-clock oscillator, use R_PG_Clock_Start_SUB or R_PG_Clock_Stop_SUB.

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();
}
```

4.1.2 R_PG_Clock_Start_SUB

Definition bool R_PG_Clock_Start_SUB(void)

Description Start the sub-clock oscillator

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

- This function starts the sub-clock oscillator.
- The sub-clock oscillator shall be stopped in the function R_PG_Clock_Set. To start the sub-clock oscillator after calling R_PG_Clock_Set, call this function.

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();

    //Start the sub-clock oscillator.
    R_PG_Clock_Start_SUB();
}
```

4.1.3 R_PG_Clock_Stop_SUB

Definition bool R_PG_Clock_Stop_SUB(void)

Description Stop the sub-clock oscillator

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

- This function stops the sub-clock oscillator.

Example

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

void func1(void)
{
    //Set the clock-generation circuit.
    R_PG_Clock_Set();

    //Start the sub-clock oscillator.
    R_PG_Clock_Start_SUB();
}

void func2(void)
{
    //Stop the sub-clock oscillator.
    R_PG_Clock_Stop_SUB();
}
```

4.1.4 R_PG_Clock_GetMainClockStatus

Definition: bool R_PG_Clock_GetMainClockStatus(bool * stop)

Description: Get the main clock oscillation stop detection flag

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

Parameter

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

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output: R_PG_Clock.c

RPDL function: R_CGC_GetStatus

Details

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

Example

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

bool main_stop;

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

4.2 Voltage Detection Circuit (LVD)

4.2.1 R_PG_LVD_Set

Definition: bool R_PG_LVD_Set (void)

Description: Set up the voltage detection circuit.

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_LVD.c

RPDL function: R_LVD_Control

Details:

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

Example:

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

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

4.2.2 R_PG_LVD_GetLVDDetectionFlag

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

Description: Acquire the value of the LVD detection flags.

Parameter

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_LVD.c

RPDL function: R_LPC_GetStatus

Details

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

Example

```
//Include "R_PG_<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_LVD_GetLVDDetectionFlag ( &lvd1, &lvd2);

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

4.3 Low Power Consumption

4.3.1 R_PG_LPC_Set

Definition: bool R_PG_LPC_Set (void)

Description: Set up the low power consumption functions.

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_LPC.c

RPDL function: R_LPC_Create

Details:

- This function configures the low power conditions.

Example:

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

4.3.2 R_PG_LPC_Sleep

Definition bool R_PG_LPC_Sleep (void)

Description Enter sleep mode.

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_LPC.c

RPDL function R_LPC_Control

Details

- This function set the system to sleep mode.

Example

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

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

4.3.3 R_PG_LPC_AllModuleClockStop

Definition bool R_PG_LPC_AllModuleClockStop (void)

Description Enter all module clock stop mode.

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_LPC.c

RPDL function R_LPC_Control

Details

- This function sets the system to all module clock stop mode.
- Before entering all module clock stop mode, this function sets TMR unit which is allowed to operate while all module clock stop mode.
- By default, TMR stops while the MCU is in all module clock stop mode. To prevent stopping TMR in all module clock stop mode, select the TMR unit that you wish to operate through the GUI.

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();
}
```

4.3.4 R_PG_LPC_SoftwareStandby

Definition bool R_PG_LPC_SoftwareStandby(void)

Description Enter software standby mode.

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_LPC.c

RPDL function R_LPC_Control

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

Example

```
//Include "R_PG_<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();

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

4.3.5 R_PG_LPC_DeepSoftwareStandby

Definition bool R_PG_LPC_DeepSoftwareStandby(void)

Description Enter deep software standby mode.

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_LPC.c

RPDL function R_LPC_Control

Details

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

Example

```
//Include "R_PG_<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();

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

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

4.3.6 R_PG_LPC_IOPortRelease

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

Return value

true	Setting was made correctly
false	Setting failed

File for output	R_PG_LPC.c
RPDL function	R_LPC_Control

Details

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

Example

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

4.3.7 R_PG_LPC_GetPowerOnResetFlag

Definition: bool R_PG_LPC_GetPowerOnResetFlag (bool *reset)

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

Parameter

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

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output: R_PG_LPC.c

RPDL function: R_LPC_GetStatus

Details

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

Example

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

4.3.8 R_PG_LPC_GetLVDDetectionFlag

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

Description: Acquire the value of the LVD detection flags.

Parameter

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

Example

```
//Include "R_PG_<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 ( &lvd1, &lvd2);

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

4.3.9 R_PG_LPC_GetDeepSoftwareStandbyResetFlag

Definition: bool R_PG_LPC_GetDeepSoftwareStandbyResetFlag(bool *reset)

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

Parameter

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

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output: R_PG_LPC.c

RPDL function: R_LPC_GetStatus

Details

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

Example

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

4.3.10 R_PD_LPC_GetDeepSoftwareStandbyCancelFlag

Definition

bool R_PD_LPC_GetDeepSoftwareStandbyCancelFlag
(bool *irq0, bool *irq1, bool *irq2, bool *irq3, bool *lvd, bool *rtc, bool *usb, bool *nmi)

Description

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

Parameter

bool *irq0	The address of storage area for the flag of cancel request by IRQ0
bool *irq1	The address of storage area for the flag of cancel request by IRQ1
bool *irq2	The address of storage area for the flag of cancel request by IRQ2
bool *irq3	The address of storage area for the flag of cancel request by IRQ3
bool *lvd	The address of storage area for the flag of cancel request by LVD
bool *rtc	The address of storage area for the flag of cancel request by RTC
bool *usb	The address of storage area for the flag of cancel request by USB
bool *nmi	The address of storage area for the flag of cancel request by NMI

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output

R_PG_LPC.c

RPDL function

R_LPC_GetStatus

Details

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

Example

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

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

4.3.11 R_PG_LPC_GetStatus

Definition: bool R_PG_LPC_GetStatus(uint16_t *data)

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

Parameter

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

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output: R_PG_LPC.h

RPDL function: R_LPC_GetStatus

Details

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

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

b7	b6	b5	b4	b3	b2	b1	b0
Deep software standby cancel request detection (DPSIFR) (0: not detected; 1: detected)							
NMI	USB suspend/ resume	RTC	LVD	IRQ3-A	IRQ2-A	IRQ1-A	IRQ0-A

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

Example

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

4.3.12 R_PG_LPC_WriteBackup

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

Description: Write data into the deep standby backup registers.

Parameter

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

Return value

true	Setting was made correctly
false	Setting failed

File for output: R_PG_LPC.h

RPDL function: R_LPC_WriteBackup

Details

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

Example

```
//Include "R_PG_<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();

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

4.3.13 R_PG_LPC_ReadBackup

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

Description: Read data from the deep standby backup registers.

Parameter

uint8_t * data	The start address of 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();

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

4.4 Interrupt Controller (ICUa)

4.4.1 R_PG_ExtInterrupt_Set_<*interrupt type*>

Definition: bool R_PG_ExtInterrupt_Set_<*interrupt type*> (void)
<*interrupt type*>: IRQ0 to IRQ15 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 IRQ15 or NMI

RPDL function: R_INTC_CreateExtInterrupt

Details:

- Enables an external interrupt (IRQ0 to IRQ15 or NMI) and sets the input direction and input buffer for the pins to be used for the external interrupt signal. For IRQn, the pin to be used (IRQn-A/B) 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*>.

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();
}
```

4.4.2 R_PG_ExtInterrupt_Disable_<*interrupt type*>

Definition bool R_PG_ExtInterrupt_Disable_<*interrupt type*> (void)
<*interrupt type*>: IRQ0 to IRQ15

Description Disable an external interrupt

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_ExtInterrupt_<*interrupt type*>.c
<*interrupt type*>: IRQ0 to IRQ15

RPDL function R_INTC_ControlExtInterrupt

Details
- Disables an external interrupt (IRQ0 to IRQ15).
- Settings of the input/output direction and input buffer for the pin being used for the external interrupt signal are retained.

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

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

4.4.3 R_PG_ExtInterrupt_GetRequestFlag_<*interrupt type*>

Definition

bool R_PG_ExtInterrupt_GetRequestFlag_<*interrupt type*> (bool * flag)

<*interrupt type*>: IRQ0 to IRQ15 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 IRQ15 or NMI

RPDL function

R_INTC_GetExtInterruptStatus

Details

- Acquires the interrupt request flag for an external interrupt (IRQ0 to IRQ15 or NMI). When an interrupt is requested, ‘true’ is entered in the specified destination for storage of the flag’s value.

Example

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

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

4.4.4 R_PG_ExtInterrupt_ClearRequestFlag_<*interrupt type*>

Definition bool R_PG_ExtInterrupt_ClearRequestFlag_<*interrupt type*> (void)
<*interrupt type*>: IRQ0 to IRQ15 or NMI

Description Clear an external interrupt request flag

Parameter None

Return value

true	Clearing succeeded
false	Clearing failed

File for output R_PG_ExtInterrupt_<*interrupt type*>.c
<*interrupt type*>: IRQ0 to IRQ15 or NMI

RPDL function R_INTC_ControlExtInterrupt

Details
- Clears the interrupt request flag for an external interrupt (IRQ0 to IRQ15 or NMI).
- If the level-sensitive interrupt is selected, the interrupt request flag is cleared when high-level is input to the interrupt pin. The request flag of level-sensitive interrupt cannot be cleared by this function.

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

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

4.4.5 R_PG_SoftwareInterrupt_Set

Definition: bool R_PG_SoftwareInterrupt_Set(void)

Description: Set up the software interrupt

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_SoftwareInterrupt.c

RPDL function: R_INTC_CreateSoftwareInterrupt

Details:

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

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

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

4.4.6 R_PG_SoftwareInterrupt_Generate

Definition: bool R_PG_SoftwareInterrupt_Generate(void)

Description: Generate the software interrupt

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_SoftwareInterrupt.c

RPDL function: R_INTC_Write

Details:

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

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

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

4.4.7 R_PG_FastInterrupt_Set

Definition bool R_PG_FastInterrupt_Set (void)

Description Set up the fast interrupt

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_FastInterrupt.c

RPDL function R_INTC_CreateFastInterrupt

Details

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

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

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

4.4.8 R_PG_Exception_Set

Definition: bool R_PG_Exception_Set (void)

Description: Set the exception handlers

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Exception.c

RPDL function: R_INTC_CreateExceptionHandlers

Details:

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

```
//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
}

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

4.5 Buses

4.5.1 R_PG_ExtBus_SetBus

Definition: bool R_PG_ExtBus_SetBus(void)

Description: Set up the bus pins and 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 pins and the bus error monitoring.
- The bus error interrupt is set by this function. If [Notify the bus error interrupt by function call] is selected in the GUI, the function having the specified name will be called when an interrupt occurs. Create the interrupt notification function as follows:
  void <*name of the interrupt notification function*> (void)
  For the interrupt notification function, note the contents of this chapter end, Notes on Notification Functions.
- The status of bus error generation can be acquired by calling R_PG_ExtBus_GetErrorStatus.
- The pin assignment (-A/-B/-C) is set according to the selection in the [Peripheral Pin Usage] window.
- The external bus clock (BCLK) and SDRAM clock (SDCLK) can be set by R_PG_Clock_Set.

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();
}
```

4.5.2 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

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 6:EDMAC 7:EXDMAC
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();
}
```

4.5.3 R_PG_ExtBus_ClearErrorFlags

Definition: bool R_PG_ExtBus_ClearErrorFlags(void)

Description: Clear the bus-error status registers

Parameter: None

Return value:

true	Clearing succeeded
false	Clearing failed

File for output: R_PG_ExtBus.c

RPDL function: R_BSC_Control

Details:

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

Example: 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();
}
```

4.5.4 R_PG_ExtBus_SetArea_CS<*CS area number*>

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

<*CS area number*>: 0 to 7

Description: Set up CS area

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_ExtBus_CS<area number>.c

<*CS area number*>: 0 to 7

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 CS0, CS6 and SDRAM area (SDCS) 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 CS0
    R_PG_ExtBus_SetArea_CS0();

    //Set up CS6
    R_PG_ExtBus_SetArea_CS6();

    //Set up SDRAM area (SDCS)
    R_PG_ExtBus_SetArea_SDCS();
}
```

4.5.5 R_PG_ExtBus_DisableArea_CS<*CS area number*>

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

Description: Disable CS area

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 7

RPDL function: R_BSC_Destroy

Details:

- Disables 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();

    //Set up CS6
    R_PG_ExtBus_SetArea_CS6();

    //Set up SDRAM area (SDCS)
    R_PG_ExtBus_SetArea_SDCS();
}

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

    //Disable CS6
    R_PG_ExtBus_DisableArea_CS6();

    //Disable SDRAM area (SDCS)
    R_PG_ExtBus_DisableArea_SDCS();
}
```

4.5.6 R_PG_ExtBus_SetArea_SDCS

Definition: bool R_PG_ExtBus_SetArea_SDCS(void)

Description: Set up SDRAM area (SDCS)

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_ExtBus_SDCS.c

RPDL function: R_BSC_SDRAM_CreateArea

Details:

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

Example: A case where CS0, CS6 and SDRAM area (SDCS) 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 CS0
    R_PG_ExtBus_SetArea_CS0();

    //Set up CS6
    R_PG_ExtBus_SetArea_CS6();

    //Set up SDRAM area (SDCS)
    R_PG_ExtBus_SetArea_SDCS();
}
```

4.5.7 R_PG_ExtBus_Initialize_SDCS

Definition: bool R_PG_ExtBus_Initialize_SDCS(void)

Description: Start the SDRAM initialization sequence

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_ExtBus_SDCS.c

RPDL function: R_BSC_Control

Details:

- Starts the SDRAM initialization sequence.
- The initialization sequence must be started when the SDRAM operation, the auto refresh, and the self refresh are disabled.

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 bus pins and bus error monitoring.
    R_PG_ExtBus_SetBus();

    //Set up SDRAM area (SDCS)
    R_PG_ExtBus_SetArea_SDCS();

    //Start initialization sequence
    R_PG_ExtBus_Initialize_SDCS();

    //Enable auto refresh
    R_PG_ExtBus_AutoRefreshEnable_SDCS();

    //Enable SDRAM operation
    R_PG_ExtBus_AccessEnable_SDCS();
}
```

4.5.8 R_PG_ExtBus_AutoRefreshEnable_SDCS

Definition: bool R_PG_ExtBus_AutoRefreshEnable_SDCS(void)

Description: Enable the SDRAM auto refresh

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_ExtBus_SDCS.c

RPDL function: R_BSC_Control

Details:

- Enables the SDRAM auto refresh.
- The SDRAM auto refresh must be started when the SDRAM self refresh is disabled.

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();

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

    //Set up SDRAM area (SDCS)
    R_PG_ExtBus_SetArea_SDCS();

    //Start SDRAM initialization sequence
    R_PG_ExtBus_Initialize_SDCS();

    //Enable SDRAM auto refresh
    R_PG_ExtBus_AutoRefreshEnable_SDCS();

    //Enable SDRAM operation
    R_PG_ExtBus_AccessEnable_SDCS();
}

void func2(void)
{
    //Disable SDRAM operation
    R_PG_ExtBus_AccessDisable_SDCS();

    //Disable SDRAM auto refresh
    R_PG_ExtBus_AutoRefreshDisable_SDCS();
}
```

4.5.9 R_PG_ExtBus_AutoRefreshDisable_SDCS

Definition: bool R_PG_ExtBus_AutoRefreshDisable_SDCS(void)

Description: Disable the SDRAM auto refresh

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_ExtBus_SDCS.c

RPDL function: R_BSC_Control

Details:

- Disables the SDRAM auto refresh.
  The SDRAM auto refresh must be stopped when the SDRAM self refresh is disabled.

Example: Refer to the example of R_PG_ExtBus_AutoRefreshEnable_SDCS.

4.5.10 R_PG_ExtBus_SelfRefreshEnable_SDCS

Definition: bool R_PG_ExtBus_SelfRefreshEnable_SDCS(void)

Description: Enable the SDRAM self refresh

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_ExtBus_SDCS.c

RPDL function: R_BSC_Control

Details:

- Enables the SDRAM self refresh
- The SDRAM self refresh mode must be started when the SDRAM operation is disabled and the auto refresh is enabled.

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 bus pins and bus error monitoring.
    R_PG_ExtBus_SetBus();

    //Set up SDRAM area (SDCS)
    R_PG_ExtBus_SetArea_SDCS();

    //Start SDRAM initialization sequence
    R_PG_ExtBus_Initialize_SDCS();

    //Enable SDRAM auto refresh
    R_PG_ExtBus_AutoRefreshEnable_SDCS();

    //Enable SDRAM operation
    R_PG_ExtBus_AccessEnable_SDCS();
}

void func2(void)
{
    //Disable SDRAM operation
    R_PG_ExtBus_AccessDisable_SDCS();

    //Enable SDRAM self refresh
    R_PG_ExtBus_SelfRefreshEnable_SDCS();
}

void func3(void)
{
    //Disable SDRAM self refresh
    R_PG_ExtBus_SelfRefreshDisable_SDCS();

    //Enable SDRAM operation
    R_PG_ExtBus_AccessEnable_SDCS();
}
```

]

4.5.11 R_PG_ExtBus_SelfRefreshDisable_SDCS

Definition: bool R_PG_ExtBus_SelfRefreshDisable_SDCS(void)

Description: Disable the SDRAM self refresh

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_ExtBus_SDCS.c

RPDL function: R_BSC_Control

Details:

- Disables the SDRAM self refresh

Example: Refer to the example of R_PG_ExtBus_SelfRefreshEnable_SDCS.

4.5.12 R_PG_ExtBus_AccessEnable_SDCS

Definition: bool R_PG_ExtBus_AccessEnable_SDCS(void)

Description: Enable SDRAM operation

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_ExtBus_SDCS.c

RPDL function: R_BSC_Control

Details:

- Enables SDRAM operation.

Example: Refer to the example of R_PG_ExtBus_AutoRefreshEnable_SDCS and R_PG_ExtBus_SelfRefreshEnable_SDCS.

4.5.13 R_PG_ExtBus_AccessDisable_SDCS

Definition: bool R_PG_ExtBus_AccessDisable_SDCS(void)

Description: Disable SDRAM operation

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_ExtBus_SDCS.c

RPDL function: R_BSC_Control

Details:

- Disables SDRAM operation.

Example: Refer to the example of R_PG_ExtBus_AutoRefreshEnable_SDCS and R_PG_ExtBus_SelfRefreshEnable_SDCS.

4.5.14 R_PG_ExtBus_GetStatus_SDCS

Definition

bool R_PG_ExtBus_GetStatus_SDCS
(bool * mode_setting, bool * initializing, bool * rec_trans)

Description

Acquire the status of SDRAM

Parameter

Parameter

bool * mode_setting	The address of storage area for the mode register setting status bit (1: Mode register setting in progress)
bool * initializing	The address of storage area for initialization status bit (1: Initialization sequence in progress)
bool * rec_trans	The address of storage area for Self-refresh transition/recovery status bit (1: Transition/recovery in progress)

true	Acquisition succeeded
false	Acquisition failed

File for output

R_PG_ExtBus.c

RPDL function

R_BSC_GetStatus

Details

- Acquire the status of SDRAM from the SDRAM SDRAM status register.
  Specify the address of storage area for the items to be acquired. Specify 0 for an item that is not required.

Example

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

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

bool mode_setting, initializing, rec_trans;

//The bus error interrupt notification function
void BusErrFunc(void)
{
    // Acquire the status of SDRAM
    R_PG_ExtBus_GetStatus_SDCS( &mode_setting, &initializing, &rec_trans );
}
```

4.6 DMA controller (DMACA)

4.6.1 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*> 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 SCIa transmission data by DMAC, make the following settings.

  DMAC settings

Transfer request source	: TXI (SCIa 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 serial transmit data register (TDR)
Destination address update mode	: Fixed
Length of a single data	: 1 byte

  SCIa setting

Data transmission method	: Transfer the transmitted serial data by DMAC

  For usage of function, refer to example 2.

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

  DMAC settings

Transfer request source	: RXI (SCIa 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 serial receive data register (RDR)
Source address update mode	: Fixed
Length of a single data	: 1 byte

  SCIa 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)
{
      //Set up DMAC0
       R_PG_DMAC_Set_C0();

      //Set up 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 DMAC
       R_PG_DMAC_StopModule_C0();
}
```

Example 2 A case where the SCIa 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.

//DMA transfer end flag
volatile bool sci_dma_transfer_complete;

//Data source
uint8_t tr[]=”ABCDEFG”;

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

	//Set up SCI0
	R_PG_SCI_Set_C0();

	//Set up DMAC0
	R_PG_DMAC_Set_C0();

	//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)
{
	//SCIa transmit end flag
	bool sci_transfer_complete;
	sci_transfer_complete = false;

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

	//Stop the SCIa
	R_PG_SCI_StopCommunication_C0();

	//Stop the DMAC
	R_PG_DMAC_StopModule_C0();

	sci_dma_transfer_complete = true;
}
```

Example 3 A case where the SCIa 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 seleclted as a DMA transfer trigger.

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

//Data destination
uint8_t re[]="--------";

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

      //Set up SCI0
      R_PG_SCI_Set_C0();

      //Set up DMAC0
      R_PG_DMAC_Set_C0();

      //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 SCIa reception
      R_PG_SCI_StopCommunication_C0();

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

4.6.2 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();
}
```

4.6.3 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 data transfer (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 triggers the DMA transfer.
- This function is genetarted when the software trigger 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.
- 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;
}
```

4.6.4 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 the DMA transfer.
- 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
- Irq1ExtIntFunc was specified as the IRQ1 interrupt notification function name
- Irq2ExtIntFunc 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 Irq1ExtIntFunc (void)
{
      R_PG_DMAC_Suspend_C0();       //Suspend the DMA transfer
}

//DMA transfer is re-activated by IRQ2 input
void Irq2ExtIntFunc (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
}
```

4.6.5 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();
}
```

4.6.6 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 written 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();
}
```

4.6.7 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();
}
```

4.6.8 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 written 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();
}
```

4.6.9 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 );
}
```

4.6.10 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();
}
```

4.6.11 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();
}
```

4.6.12 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();
}
```

4.6.13 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();
}
```

4.6.14 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();
}
```

4.6.15 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();
}
```

4.6.16 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();
}
```

4.6.17 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();
}
```

4.6.18 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();
}
```

4.6.19 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();
}
```

4.7 EXDMAC controller (EXDMAC)

4.7.1 R_PG_EXDMAC_Set_C<*channel number*>

Definition

bool R_PG_EXDMAC_Set_C<*channel number*> (void)
<*channel number*>: 0, 1

Description

Set up an EXDMAC channel

Parameter

None

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_EXDMAC_C <*channel number*>.c
<*unit number*>: 0, 1

RPDL function

R_EXDMAC_Create

Details

- Releases the EXDMAC from the module-stop and makes initial settings.
- If the transfer start trigger other than the software trigger was selected, the EXDMAC channel will be ready for the trigger signal by calling R_PG_EXDMAC_Activate_C<*channel number*> after calling this function. If the software trigger was selected as a transfer start trigger, EXDMAC channel will start the data transfer when calling R_PG_EXDMAC_StartTransfer_C<*channel number*> after calling this function.
- If the external signal was selected as a transfer start trigger or if EDACK signal was enabled, this function sets the pins to be used. The pins used for external signal input or EDACK output can be selected through the Peripheral Pin Usage window in GUI.
- The EXDMAC 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.

Example

A case where the setting is made as follows.

- EDREQ0 signal or MTU was selected as a transfer start trigger of EXDMAC0.
- Exdmac0IntFunc was specified as the EXDMAC 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 EXDMAC0
      R_PG_EXDMAC_Set_C0();

      // Make EXDMAC0 be ready for the transfer start trigger
      R_PG_EXDMAC_Activate_C0();
}

//EXDMAC interrupt notification function
void Exdmac0IntFunc (void)
{
      //Stop the EXDMAC
      R_PG_EXDMAC_StopModule_C0();
}
```

4.7.2 R_PG_EXDMAC_Activate_C<*channel number*>

Definition: bool R_PG_EXDMAC_Activate_C<*channel number*> (void)
< *channel number* > : 0, 1

Description: Make the EXDMAC be ready for the start trigger

Conditions for output: An external signal or MTU is selected as a transfer start trigger

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_EXDMAC_C <*channel number*>.c
<*channel number*>: 0, 1

RPDL function: R_EXDMAC_Control

Details:
- This function makes the EXDMAC channel be ready for the transfer start trigger.
- This function is genetarted when an external signal or MTU is selected as a transfer start trigger.
- Call R_PG_EXDMAC_Set_C<*channel number*> to set up an EXDMAC channel before calling this function.

Example: A case where the setting is made as follows.
- EDREQ0 signal or MTU was selected as a transfer start trigger of EXDMAC0.
- Exdmac0IntFunc was specified as the EXDMAC 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 EXDMAC0
    R_PG_EXDMAC_Set_C0();

    // Make EXDMAC0 be ready for the transfer start trigger
    R_PG_EXDMAC_Activate_C0();
}

//EXDMAC interrupt notification function
void Exdmac0IntFunc (void)
{
    //Stop the EXDMAC
    R_PG_EXDMAC_StopModule_C0();
}
```

4.7.3 R_PG_EXDMAC_StartTransfer_C<*channel number*>

Definition: bool R_PG_EXDMAC_StartTransfer_C<*channel number*> (void)
< *channel number* > : 0, 1

Description: Start the data transfer (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_EXDMAC_C <*channel number*>.c
<*channel number*>: 0, 1

RPDL function: R_EXDMAC_Control

Details:
- This function triggers the DMA transfer.
- This function is genetarted when the software trigger is selected as a transfer start trigger.
- Call R_PG_EXDMAC_Set_C<*channel number*> to set up an EXDMAC channel 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 EXDMAC0 in normal transfer mode
- Exdmac0IntFunc was specified as the EXDMAC 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 EXDMAC0
    R_PG_EXDMAC_Set_C0();

    while( transferred == false ){
        //Start the DMA transfer of EXDMAC0
        R_PG_EXDMAC_StartTransfer_C0();
    }
    //Stop the EXDMAC
    R_PG_EXDMAC_StopModule_C0();
}

//EXDMAC interrupt notification function
void Exdmac0IntFunc (void)
{
    transferred = true;
}
```

4.7.4 R_PG_EXDMAC_Suspend_C<*channel number*>

Definition: bool R_PG_EXDMAC_Suspend_C<*channel number*> (void)
< *channel number* > : 0, 1

Description: Suspend the data transfer

Parameter: None

Return value:

true	Suspending succeeded.
false	Suspending failed.

File for output: R_PG_EXDMAC_C <*channel number*>.c
<*channel number*>: 0, 1

RPDL function: R_EXDMAC_Control

Details:
- This function suspends the DMA transfer.
- To resume the transfer, when an external signal or MTU is selected as a transfer start trigger, call R_PG_EXDMAC_Activate_C<*channel number*>.

Example: A case where the setting is made as follows.
- EDREQ0 signal or MTU was selected as a transfer start trigger of EXDMAC0.
- Exdmac0IntFunc was specified as the EXDMAC interrupt notification function name
- Irq1ExtIntFunc was specified as the IRQ1 interrupt notification function name
- Irq2ExtIntFunc 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_EXDMAC_Set_C0();       //Set up EXDMAC0
      R_PG_ExtInterrupt_Set_IRQ0();   //Set IRQ0
      R_PG_ExtInterrupt_Set_IRQ1();   //Set IRQ1
      R_PG_ExtInterrupt_Set_IRQ2();   //Set IRQ2
      R_PG_EXDMAC_Activate_C0(); // Make EXDMAC0 be ready for the transfer start trigger
}

//EXDMAC interrupt notification function
void Exdmac0IntFunc (void)
{
      R_PG_EXDMAC_StopModule_C0();       //Stop the EXDMAC0
}

//DMA transfer is suspended by IRQ1 input
void Irq1ExtIntFunc (void)
{
      R_PG_EXDMAC_Suspend_C0();       //Suspend the DMA transfer
}

//DMA transfer is re-activated by IRQ2 input
void Irq2ExtIntFunc (void)
{
      R_PG_EXDMAC_Activate_C0(); // Make EXDMAC0 be ready for the transfer start trigger
}
```

4.7.5 R_PG_EXDMAC_GetTransferCount_C<*channel number*>

Definition

bool R_PG_EXDMAC_GetTransferCount_C<*channel number*> (uint16_t * count)

< *channel number* > : 0, 1

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_EXDMAC_C <*channel number*>.c

<*channel number*>: 0, 1

RPDL function

R_EXDMAC_GetStatus

Details

- This function gets the current transfer counter value.
- The EXDMAC interrupt request flag (IR flag) is cleared in this function. Call R_PG_EXDMAC_ClearInterruptFlag_C<*channel number*> to get the EXDMAC interrupt request flag before calling this function if needed.

Example

A case where the setting is made as follows.

- EDREQ0 signal or MTU was selected as a transfer start trigger of EXDMAC0.

```
#include "R_PG_default.h" //Include "R_PG_<PDG project name>.h" to use this function.
void func(void)
{
    uint16_t count;

    //Set up EXDMAC0
    R_PG_EXDMAC_Set_C0();

    //Make EXDMAC0 be ready for the transfer start trigger
    R_PG_EXDMAC_Activate_C0();

    //Wait for the transfer counter to become lower than 10
    do{
        R_PG_EXDMAC_GetTransferCount_C0( & count );
    } while( count >= 10 );

    //Suspend the DMA transfer
    R_PG_EXDMAC_Suspend_C0();
}
```

4.7.6 R_PG_EXDMAC_SetTransferCount_C<*channel number*>

Definition

bool R_PG_EXDMAC_SetTransferCount_C<*channel number*>(uint16_t count)

< *channel number* > : 0, 1

Description

Set the transfer counter

Parameter

uint16_t count	Value to be written to the transfer counter

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_EXDMAC_C<*channel number*>.c

<*channel number*>: 0, 1

RPDL function

R_EXDMAC_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, block transfer mode and cluster transfer mode.

Example

A case where the setting is made as follows.

- EDREQ0 signal or MTU was selected as a transfer start trigger of EXDMAC0
- Exdmac0IntFunc was specified as the EXDMAC 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 EXDMAC0
    R_PG_EXDMAC_Set_C0();

    // Make EXDMAC0 be ready for the transfer start trigger
    R_PG_EXDMAC_Activate_C0();
}
//EXDMAC interrupt notification function
void Exdmac0IntFunc (void)
{
    //Suspend the DMA transfer
    R_PG_EXDMAC_Suspend_C0();

    //Change the EXDMAC0 settings
    R_PG_EXDMAC_SetSrcAddress_C0( src_address );    //Source address
    R_PG_EXDMAC_SetDestAddress_C0( dest_address );    //Destination address
    R_PG_EXDMAC_SetTransferCount_C0( tr_count );    //Transfer counter

    // Make EXDMAC0 be ready for the transfer start trigger
    R_PG_EXDMAC_Activate_C0();
}
```

4.7.7 R_PG_EXDMAC_GetRepeatBlockSizeCount_C<*channel number*>

Definition: bool R_PG_EXDMAC_GetRepeatBlockSizeCount_C<*channel number*> (uint16_t * count)
< *channel number* > : 0, 1

Description: Get the repeat/block/cluster size counter value

Conditions for output: Repeat transfer mode, block transfer mode or cluster 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_EXDMAC_C <*channel number*>.c
<*channel number*>: 0, 1

RPDL function: R_EXDMAC_GetStatus

Details
- This function gets the current repeat/block/cluster size counter value.
- The EXDMAC interrupt request flag (IR flag) is cleared in this function. Call R_PG_EXDMAC_ClearInterruptFlag_C<*channel number*> to get the EXDMAC interrupt request flag before calling this function if needed.

Example: A case where the setting is made as follows.
- EXDMAC0 is set to repeat transfer mode
- The transfer start trigger is EDREQ0 signal or MTU

```
#include "R_PG_default.h" //Include "R_PG_<PDG project name>.h" to use this function.
void func(void)
{
    uint16_t count;

    //Set up EXDMAC0
    R_PG_EXDMAC_Set_C0();

    //Make EXDMAC0 be ready for the transfer start trigger
    R_PG_EXDMAC_Activate_C0();

    //Wait for the repeat size counter to become lower than 10
    do{
        R_PG_EXDMAC_GetRepeatBlockSizeCount_C0( & count );
    } while( count >= 10 );

    //Suspend the DMA transfer
    R_PG_EXDMAC_Suspend_C0();
}
```

4.7.8 R_PG_EXDMAC_SetRepeatBlockSizeCount_C<*channel number*>

Definition

bool R_PG_EXDMAC_SetRepeatBlockSizeCount_C<*channel number*> (uint16_t count)

< *channel number* > : 0, 1

Description

Set the repeat/block/cluster size counter value

Conditions for output

Repeat transfer mode, block transfer mode or cluster transfer mode is selected for the transfer mode.

Parameter

uint16_t count	Value to be written to the repeat/block/cluster size counter

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_EXDMAC_C <*channel number*>.c

<*channel number*>: 0, 1

RPDL function

R_EXDMAC_GetStatus

Details

- This function sets the repeat/block/cluster size counter.
  The valid range of the counter value is from 1 to 1023 in repeat transfer mode and block transfer mode, 1 to 7 in cluster transfer mode.

Example

A case where the setting is made as follows.

- EXDMAC0 is set to repeat transfer mode
- EDREQ0 signal or MTU was selected as a transfer start trigger
- Exdmac0IntFunc was specified as the EXDMAC 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 EXDMAC0
    R_PG_EXDMAC_Set_C0();

    // Make EXDMAC0 be ready for the transfer start trigger
    R_PG_EXDMAC_Activate_C0();
}
//EXDMAC interrupt notification function
void Exdmac0IntFunc (void)
{
    //Suspend the DMA transfer
    R_PG_EXDMAC_Suspend_C0();

    //Change the EXDMAC0 settings
    R_PG_EXDMAC_SetTransferCount_C0( tr_count );    //Transfer counter
    R_PG_EXDMAC_SetRepeatBlockSizeCount_C0( repeat_count );    //Repeat size counter

    // Make EXDMAC0 be ready for the transfer start trigger
    R_PG_EXDMAC_Activate_C0();
}
```

4.7.9 R_PG_EXDMAC_ClearInterruptFlag_C<*channel number*>

Definition: bool R_PG_EXDMAC_ClearInterruptFlag_C<*channel number*> (bool * int_request)
< *channel number* > : 0, 1

Description: Get and clear the interrupt request flag

Conditions for output: EXDMAC 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_EXDMAC_C <*channel number*>.c
<*channel number*>: 0, 1

RPDL function: R_EXDMAC_GetStatus

Details
- This function gets and clears the EXDMAC interrupt request flag (IR flag).

Example

A case where the setting is made as follows.
- EXDMAC0 is set to mormal transfer mode
- The transfer start trigger is EDREQ0 signal or MTU
- The EXDMAC interrupt is enabled
- The EXDMAC 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 EXDMAC0
    R_PG_EXDMAC_Set_C0();

    //Make EXDMAC0 be ready for the transfer start trigger
    R_PG_EXDMAC_Activate_C0();

    //Wait for the IR flag to become 1
    do{
        R_PG_EXDMAC_ClearInterruptFlag_C0(& int_request );
    } while( int_request == false );
}
```

4.7.10 R_PG_EXDMAC_GetTransferEndFlag_C<*channel number*>

Definition

bool R_PG_EXDMAC_GetTransferEndFlag_C<*channel number*> (bool* end)

< *channel number* > : 0, 1

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_EXDMAC_C <*channel number*>.c

<*channel number*>: 0, 1

RPDL function

R_EXDMAC_GetStatus

Details

- This function gets the transfer end flag.
- The EXDMAC interrupt request flag (IR flag) is cleared in this function. Call R_PG_EXDMAC_ClearInterruptFlag_C<*channel number*> to get the EXDMAC interrupt request flag before calling this function if needed.
- The transfer end flag is not cleared in this function. Call R_PG_EXDMAC_ClearTransferEndFlag_C<*channel number*> to clear the transfer end flag if needed.

Example

A case where the setting is made as follows.

- EXDMAC0 is set to normal transfer mode
- The transfer start trigger is EDREQ0 signal or MTU
- The EXDMAC 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 EXDMAC0
    R_PG_EXDMAC_Set_C0();

    //Make EXDMAC0 be ready for the transfer start trigger
    R_PG_EXDMAC_Activate_C0();

    //Wait for the transfer end flag to become 1
    do{
        R_PG_EXDMAC_GetTransferEndFlag_C0( & end );
    } while( end == false );

    //Clear the DMA transfer end flag
    R_PG_EXDMAC_ClearTransferEndFlag_C0();
}
```

4.7.11 R_PG_EXDMAC_ClearTransferEndFlag_C<*channel number*>

Definition: bool R_PG_EXDMAC_ClearTransferEndFlag_C<*channel number*> (void)
< *channel number* > : 0, 1

Description: Clear the transfer end flag

Parameter: None

Return value:

true	Clearing succeeded
false	Clearing failed

File for output: R_PG_EXDMAC_C <*channel number*>.c
<*channel number*>: 0, 1

RPDL function: R_EXDMAC_Control

Details:
- This function clears the transfer end flag.
- To get the transfer end flag, call R_PG_EXDMAC_GetTransferEndFlag_C<*channel number*>.

Example: A case where the setting is made as follows.
- EXDMAC0 is set to mormal transfer mode
- The transfer start trigger is EDREQ0 signal or MTU
- The EXDMAC 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 EXDMAC0
    R_PG_EXDMAC_Set_C0();

    //Make EXDMAC0 be ready for the transfer start trigger
    R_PG_EXDMAC_Activate_C0();

    //Wait for the transfer end flag to become 1
    do{
        R_PG_EXDMAC_GetTransferEndFlag_C0( & end );
    } while( end == false );

    //Clear the DMA transfer end flag
    R_PG_EXDMAC_ClearTransferEndFlag_C0();
}
```

4.7.12 R_PG_EXDMAC_GetTransferEscapeEndFlag_C<*channel number*>

Definition: bool R_PG_EXDMAC_GetTransferEscapeEndFlag_C<*channel number*> (bool* end)
< *channel number* > : 0, 1

Description: Get the transfer escape end flag

Conditions for output: [Completion of repeat/block/cluster 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_EXDMAC_C <*channel number*>.c
<*channel number*>: 0, 1

RPDL function: R_EXDMAC_GetStatus

Details

- This function gets the DMA transfer escape end flag (EDMSTS.ESIF).
- The EXDMAC interrupt request flag (IR flag) is cleared in this function. Call R_PG_EXDMAC_ClearInterruptFlag_C<*channel number*> to get the EXDMAC interrupt request flag before calling this function if needed.
- The transfer escape end flag is not cleared in this function. Call R_PG_EXDMAC_ClearTransferEscapeEndFlag_C<*channel number*> to clear the transfer escape end flag if needed.
- When using the external signal or MTU as a transfer start trigger, to continue the transfer after transfer escape end, call R_PG_EXDMAC_Activate_C<*channel number*>. In R_PG_EXDMAC_Activate_C<*channel number*>, the transfer escape end flag shall be cleared and EXDMAC shall be ready for the transfer start trigger.
- When using the software trigger as a transfer start trigger, to continue the transfer after transfer escape end, call R_PG_EXDMAC_Activate_C<*channel number*>. In R_PG_EXDMAC_Activate_C<*channel number*>, the transfer escape end flag shall be cleared and the transfer starts.

Example A case where the setting is made as follows.

- EXDMAC0 is set to repeat transfer mode
- The transfer start trigger is external signal or MTU
- [Completion of repeat/block/cluster size transfer] and [Transfer end] are selected for the interrupt output source
- Exdmac0IntFunc is specified as an EXDMAC interrupt notification function

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

void Exdmac0IntFunc(void)
{
    bool transfer_end;

    R_PG_EXDMAC_GetTransferEndFlag_C0( & transfer_end );
    if( transfer_end ){
        //Transfer end
        R_PG_EXDMAC_StopModule_C0();
    }
    else{
        //Transfer escape end (repeat size end)
        R_PG_DMAC_Activate_C0(); //Clear the transfer escape end flag and re-activate
    }
}

void func(void)
{
    //Set up EXDMAC0
    R_PG_EXDMAC_Set_C0();

    //Make EXDMAC0 be ready for the transfer start trigger
    R_PG_EXDMAC_Activate_C0();
}
```

4.7.13 R_PG_EXDMAC_ClearTransferEscapeEndFlag_C<*channel number*>

Definition: bool R_PG_EXDMAC_ClearTransferEscapeEndFlag_C<*channel number*> (void)
< *channel number* > : 0, 1

Description: Clear the transfer escape end flag

Conditions for output: [Completion of repeat/block/cluster 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_EXDMAC_C <*channel number*>.c
<*channel number*>: 0, 1

RPDL function: R_EXDMAC_Control

Details:
- This function clears the transfer escape end flag.
- To get the transfer escape end flag, call R_PG_EXDMAC_GetTransferEscapeEndFlag_C<*channel number*>.

Example: A case where the setting is made as follows.
- EXDMAC0 is set to repeat transfer mode
- The transfer start trigger is EDREQ0 signal or MTU
- [Completion of repeat/block/cluster size transfer] is selected for the interrupt output source
- The EXDMAC 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 EXDMAC0
    R_PG_EXDMAC_Set_C0();

    //Make EXDMAC0 be ready for the transfer start trigger
    R_PG_EXDMAC_Activate_C0();

    //Wait for the transfer escape end flag to become 1
    do{
        R_PG_EXDMAC_GetTransferEscapeEndFlag_C0( & end );
    } while( end == false );

    //Clear the DMA transfer escape end flag
    R_PG_EXDMAC_ClearTransferEscapeEndFlag_C0();
}
```

4.7.14 R_PG_EXDMAC_SetSrcAddress_C<*channel number*>

Definition

bool R_PG_EXDMAC_SetSrcAddress_C<*channel number*>(void * src_addr)

< *channel number* > : 0, 1

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_EXDMAC_C<*channel number*>.c

<*channel number*>: 0, 1

RPDL function

R_EXDMAC_Control

Details

- This function sets the source address.

Example

A case where the setting is made as follows.

- EDREQ0 signal or MTU was selected as a transfer start trigger of EXDMAC0
- Exdmac0IntFunc was specified as the EXDMAC 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 EXDMAC0
    R_PG_EXDMAC_Set_C0();

    // Make EXDMAC0 be ready for the transfer start trigger
    R_PG_EXDMAC_Activate_C0();
}
//EXDMAC interrupt notification function
void Exdmac0IntFunc (void)
{
    //Suspend the DMA transfer
    R_PG_EXDMAC_Suspend_C0();

    //Change the EXDMAC0 settings
    R_PG_EXDMAC_SetSrcAddress_C0( src_address );    //Source address
    R_PG_EXDMAC_SetDestAddress_C0( dest_address );    //Destination address
    R_PG_EXDMAC_SetTransferCount_C0( tr_count );    //Transfer counter

    // Make EXDMAC0 be ready for the transfer start trigger
    R_PG_EXDMAC_Activate_C0();
}
```

4.7.15 R_PG_EXDMAC_SetDestAddress_C<*channel number*>

Definition

bool R_PG_EXDMAC_SetDestAddress_C<*channel number*>(void * dest_addr)

< *channel number* > : 0, 1

Description

Set the destiation 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_EXDMAC_C<*channel number*>.c

<*channel number*>: 0, 1

RPDL function

R_EXDMAC_Control

Details

- This function sets the destination address.

Example

A case where the setting is made as follows.

- EDREQ0 signal or MTU was selected as a transfer start trigger of EXDMAC0
- Exdmac0IntFunc was specified as the EXDMAC 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 EXDMAC0
    R_PG_EXDMAC_Set_C0();

    // Make EXDMAC0 be ready for the transfer start trigger
    R_PG_EXDMAC_Activate_C0();
}
//EXDMAC interrupt notification function
void Exdmac0IntFunc (void)
{
    //Suspend the DMA transfer
    R_PG_EXDMAC_Suspend_C0();

    //Set up the EXDMAC and continue
    R_PG_EXDMAC_SetSrcAddress_C0( src_address );    //Source address
    R_PG_EXDMAC_SetDestAddress_C0( dest_address );    //Destination address
    R_PG_EXDMAC_SetTransferCount_C0( tr_count );    //Transfer counter

    // Make EXDMAC0 be ready for the transfer start trigger
    R_PG_EXDMAC_Activate_C0();
}
```

4.7.16 R_PG_EXDMAC_SetAddressOffset_C<*channel number*>

Definition

bool R_PG_EXDMAC_SetAddressOffset_C<*channel number*>(int32_t offset)

< *channel number* > : 0, 1

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_EXDMAC_C<*channel number*>.c

<*channel number*>: 0, 1

RPDL function

R_EXDMAC_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.

- EDREQ0 signal or MTU was selected as a transfer start trigger of EXDMAC0
- Exdmac0IntFunc was specified as the EXDMAC 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 EXDMAC0
    R_PG_EXDMAC_Set_C0();

    // Make EXDMAC0 be ready for the transfer start trigger
    R_PG_EXDMAC_Activate_C0();
}
//EXDMAC interrupt notification function
void Exdmac0IntFunc (void)
{
    //Suspend the DMA transfer
    R_PG_EXDMAC_Suspend_C0();

    //Set up the EXDMAC and continue
    R_PG_EXDMAC_SetSrcAddress_C0( src_address );    //Source address
    R_PG_EXDMAC_SetDestAddress_C0( dest_address );    //Destination address
    R_PG_EXDMAC_SetTransferCount_C0( tr_count );    //Transfer counter
    R_PG_EXDMAC_SetAddressOffset_C0( offset );    //Address offset

    // Make EXDMAC0 be ready for the transfer start trigger
    R_PG_EXDMAC_Activate_C0();
}
```

4.7.17 R_PG_EXDMAC_SetExtendedRepeatSrc_C<*channel number*>

Definition

bool R_PG_EXDMAC_SetExtendedRepeatSrc_C<*channel number*>(uint32_t area)

< *channel number* > : 0, 1

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_EXDMAC_C<*channel number*>.c

<*channel number*>: 0, 1

RPDL function

R_EXDMAC_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.

- EDREQ0 signal or MTU was selected as a transfer start trigger of EXDMAC0
- Exdmac0IntFunc was specified as the EXDMAC 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 EXDMAC0
    R_PG_EXDMAC_Set_C0();

    // Make EXDMAC0 be ready for the transfer start trigger
    R_PG_EXDMAC_Activate_C0();
}
//EXDMAC interrupt notification function
void Exdmac0IntFunc (void)
{
    //Suspend the DMA transfer
    R_PG_EXDMAC_Suspend_C0();

    //Change the EXDMAC0 settings
    R_PG_EXDMAC_SetSrcAddress_C0( src_address );    //Source address
    R_PG_EXDMAC_SetDestAddress_C0( dest_address );    //Destination address
    R_PG_EXDMAC_SetTransferCount_C0( tr_count );    //Transfer counter
    R_PG_EXDMAC_SetExtendedRepeatSrc_C0( src_repeat );    //Source extended repeat size
    R_PG_EXDMAC_SetExtendedRepeatDest_C0( dest_repeat ); //Destination extended repeat size

    // Make EXDMAC0 be ready for the transfer start trigger
    R_PG_EXDMAC_Activate_C0();
}
```

4.7.18 R_PG_EXDMAC_SetExtendedRepeatDest_C<*channel number*>

Definition: bool R_PG_EXDMAC_SetExtendedRepeatDest_C<*channel number*>(uint32_t area)
< *channel number* > : 0, 1

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_EXDMAC_C<*channel number*>.c
<*channel number*>: 0, 1

RPDL function: R_EXDMAC_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.
- EDREQ0 signal or MTU was selected as a transfer start trigger of EXDMAC0
- Exdmac0IntFunc was specified as the EXDMAC 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 EXDMAC0
    R_PG_EXDMAC_Set_C0();

    // Make EXDMAC0 be ready for the transfer start trigger
    R_PG_EXDMAC_Activate_C0();
}
//EXDMAC interrupt notification function
void Exdmac0IntFunc (void)
{
    //Suspend the DMA transfer
    R_PG_EXDMAC_Suspend_C0();

    //Change the EXDMAC0 settings
    R_PG_EXDMAC_SetSrcAddress_C0( src_address );    //Source address
    R_PG_EXDMAC_SetDestAddress_C0( dest_address );    //Destination address
    R_PG_EXDMAC_SetTransferCount_C0( tr_count );    //Transfer counter
    R_PG_EXDMAC_SetExtendedRepeatSrc_C0( src_repeat );    //Source extended repeat size
    R_PG_EXDMAC_SetExtendedRepeatDest_C0( dest_repeat );    //Destination extended repeat
size

    // Make EXDMAC0 be ready for the transfer start trigger
    R_PG_EXDMAC_Activate_C0();
}
```

4.7.19 R_PG_EXDMAC_StopModule_C<*channel number*>

Definition bool R_PG_EXDMAC_StopModule_C<*channel number*> (void)
< *channel number* > : 0, 1

Description Stop the EXDMAC channel

Parameter None

Return value

true	Stopping succeeded.
false	Stopping failed.

File for output R_PG_EXDMAC_C<*channel number*>.c
<*channel number*>: 0, 1

RPDL function R_EXDMAC_Destroy

Details
- Stops the EXDMAC channel.
- If all EXDMAC channels are stopped, EXDMAC shall be module-stop state.
- If the MTU is being used to trigger EXDMAC transfer, stop the trigger sources before calling this function.

Example A case where the setting is made as follows.
- The MTU was selected as a transfer start trigger of EXDMAC0
- Exdmac0IntFunc was specified as the EXDMAC 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 EXDMAC0
    R_PG_EXDMAC_Set_C0();
    //Set up MTU and start the count
    R_PG_Timer_Set_MTU_U0_C1();
    R_PG_Timer_StartCount_MTU_U0_C1();
    // Make EXDMAC0 be ready for the transfer start trigger
    R_PG_EXDMAC_Activate_C0();
}
//EXDMAC interrupt notification function
void Exdmac0IntFunc (void)
{
    //Stop the MTU
    R_PG_Timer_StopModule_MTU_U0();
    //Stop the EXDMAC0
    R_PG_EXDMAC_StopModule_C0();
}
```

4.8 Data Transfer Controller (DTCa)

4.8.1 R_PG_DTC_Set

Definition: bool R_PG_DTC_Set (void)

Description: Set the common options for DTC

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Dtc.c

RPDL function: R_DTC_Set

Details:

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

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

- The DTC vector table address has been set to 15000h.
- The transfer setting of which the transfer start trigger is IRQ8 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 = 0x00015000
uint32_t dtc_vector_table [256];

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

    //Make the transfer setting of which the transfer start trigger is IRQ8
    R_PG_DTC_Set_IRQ8();

    //Make DTC be ready for the transfer start trigger
    R_PG_DTC_Activate_C0();

    //Set up IRQ8
    R_PG_ExtInterrupt_Set_IRQ8();
}
```

4.8.2 R_PG_DTC_Set_<trigger source>

Definition

bool R_PG_DTC_Set_<trigger source> (void)

< trigger source > :
SWINT, CMT0 to 3, D0FIFO0 to 1, D1FIFO0 to 1, SPRI0 to 1, SPTI0 to 1, IRQ0 to 15, ADI0 to 1, S12ADI0, TGI0A to 10D, TGIU5 to TGIW11, CMIA0 to B3, DMACIA0 to 3, EXDMACI0 to 1, RXI0 to 6, TXI0 to 6, ICRXI0 to 1, ICTXI0 to 1

Description

Set up DTC transfer data

Parameter

None

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_Dtc.c

RPDL function

R_DTC_Create

Details

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

Example

A case where the setting is made as follows.

- The DTC vector table address has been set to 15000h.
- The transfer setting of which the transfer start trigger is IRQ8 has been made.
- The transfer setting of which the transfer start trigger is IRQ9 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 = 0x00015000
uint32_t dtc_vector_table [256];

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

    //Make the transfer setting of which the transfer start trigger is IRQ8
    R_PG_DTC_Set_IRQ8();

    //Make the transfer setting of which the transfer start trigger is IRQ9
    R_PG_DTC_Set_IRQ9();

    //Make DTC be ready for the transfer start trigger
    R_PG_DTC_Activate_C0();

    //Set up IRQ8 and IRQ9
    R_PG_ExtInterrupt_Set_IRQ8();
    R_PG_ExtInterrupt_Set_IRQ9();
}
```

4.8.3 R_PG_DTC_Activate

Definition: bool R_PG_DTC_Activate (void)

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

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Dtc.c

RPDL function: R_DTC_Control

Details:

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

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

- The DTC vector table address has been set to 15000h.
- The transfer setting of which the transfer start trigger is IRQ8 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.
- Irq8IntFunc has been specified as an IRQ8 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 = 0x00015000
uint32_t dtc_vector_table [256];

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

    //Make the transfer setting of which the transfer start trigger is IRQ8
    R_PG_DTC_Set_IRQ8();

    //Make DTC be ready for the transfer start trigger
    R_PG_DTC_Activate_C0();

}

void Irq8IntFunc(void)
{
    //Disable the IRQ8
    //(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_IRQ8();
}
```

4.8.4 R_PG_DTC_SuspendTransfer

Definition bool R_PG_DTC_SuspendTransfer (void)

Description Stop the data transfer

Parameter None

Return value

true	Stopping succeeded
false	Stopping failed

File for output R_PG_Dtc.c

RPDL function R_DTC_Control

Details

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

Example A case where the setting is made as follows.

- The DTC vector table address has been set to 15000h.
- The transfer setting of which the transfer start trigger is IRQ8 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 = 0x00015000
uint32_t dtc_vector_table [256];

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

      //Make the transfer setting of which the transfer start trigger is IRQ8
      R_PG_DTC_Set_IRQ8();

      //Make DTC be ready for the transfer start trigger
      R_PG_DTC_Activate_C0();

      //Set up IRQ8
      R_PG_ExtInterrupt_Set_IRQ8();
}

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

//Resume the DTC transfer
void func3(void)
{
      R_PG_DTC_Activate_C0();
}
```

4.8.5 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 72:
                //Processing when the transfer of vector 72 is in progress
                break;
            case 73:
                //Processing when the transfer of vector 73 is in progress
                break;
            default:
        }
    }
}
```

4.8.6 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 shuts down DTC and DMAC. Use R_PG_DTC_SuspendTransfer to stop only DTC.

Example A case where the setting is made as follows.
- The DTC vector table address has been set to 15000h.
- The transfer setting of which the transfer start trigger is IRQ8 has been made.
- The transfer setting of which the transfer start trigger is IRQ9 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 = 0x00015000
uint32_t dtc_vector_table [256];

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

    //Make the transfer setting of which the transfer start trigger is IRQ8
    R_PG_DTC_Set_IRQ8();

    //Make the transfer setting of which the transfer start trigger is IRQ9
    R_PG_DTC_Set_IRQ9();

    //Make DTC be ready for the transfer start trigger
    R_PG_DTC_Activate_C0();

    //Set up IRQ8 and IRQ9
    R_PG_ExtInterrupt_Set_IRQ8();
    R_PG_ExtInterrupt_Set_IRQ9();
}

void func2(void)
{
    //Disable IRQ8 and IRQ9
    R_PG_ExtInterrupt_Disable_IRQ8();
    R_PG_ExtInterrupt_Disable_IRQ9();
    //Shut down the DTC
    R_PG_DTC_StopModule();
}
```

4.9 I/O Ports

4.9.1 R_PG_IO_PORT_Set_P<*port number*>

Definition: bool R_PG_IO_PORT_Set_P<*port number*> (void)
<*port number*>: 0 to 9 and A to G

Description: Set up the I/O port

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_IO_PORT_P<*port number*>.c
<*port number*>: 0 to 9 and A to G

RPDL function: R_IO_PORT_Set

Details:

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

Example:

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

void func(void)
{
    //Set P0.
    R_PG_IO_PORT_Set_P0();
}
```

4.9.2 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 and A to G
<*pin number*>: 0 to 7

Description: Set up the I/O port pin

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_IO_PORT_P<*port number*>.c
<*port number*>: 0 to 9 and A to G

RPDL function: R_IO_PORT_Set

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

Example:

```
//Include "R_PG_<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();

    //Set P02.
    R_PG_IO_PORT_Set_P02();
}
```

4.9.3 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 and A to G

Description

Read data from the I/O port register

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 and A to G

RPDL function

R_IO_PORT_Read

Details

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

Example

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

void func(void)
{
    uint8_t data

    //Acquire the states of P0 pins.
    R_PG_IO_PORT_Read_P0( &data );
}
```

4.9.4 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 and A to G

<*pin number*>: 0 to 7

Description

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

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 and A to G)

RPDL function

R_IO_PORT_Read

Details

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

Example

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

void func(void)
{
    uint8_t data_p00, data_p01, data_p02;

    //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);

    //Acquire the state of pin P02.
    R_PG_IO_PORT_Read_P02( & data_p02);
}
```

4.9.5 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 9 and A to G

Description

Write data to the I/O port data register

Parameter

uint8_t data	Value to be written

Return value

true	Writing proceeded correctly.
false	Writing failed.

File for output

R_PG_IO_PORT_P<*port number*>.c

<*port number*>: 0 to 9 and A to G

RPDL function

R_IO_PORT_Write

Details

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

Example

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

4.9.6 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 9 and A to G

<*pin number*>: 0 to 7

Description

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

Parameter

uint8_t data	Value to be written

Return value

true	Writing proceeded correctly.
false	Writing failed.

File for output

R_PG_IO_PORT_P<*port number*>.c

<*port number*>: 0 to 9 and A to G

RPDL function

R_IO_PORT_Write

Details

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

Example

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

4.10 Multi-Function Timer Pulse Unit 2 (MTU2)

4.10.1 R_PG_Timer_Set_MTU_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_Set_MTU_U<*unit number*>_C<*channel number*> (void)
<*unit number*>: 0 or 1
<*channel number*>: 0 to 11

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 or 1
<*channel number*>: 0 to 11

RPDL function

R_MTU2_Set, R_MTU2_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 this chapter end, Notes on Notification Functions.
- If the interrupt propriety level is set to 0 in the GUI, a CPU interrupt does not occur. The state of a request flag can be acquired by calling R_PG_Timer_GetRequestFlag_MTU_U<*unit number*>_C<*channel number*>.
- When counting driven by an externally input clock, the external reset signal, input capture, or pulse output is in use, the direction (input or output) and input buffer for the pin to be used is set in this function.
- R_PG_Timer_StartCount_MTU_U<*unit number*>_C<*channel number*>(_<*phase*>) can be used to start the count operation.

Example

A case where the setting is made as follows.

- MTU unit 1 channel 6 was set up
- 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_U1_C6();       //Set up the MTU6
    R_PG_Timer_StartCount_MTU_U1_C6();       // Start the count operation
}

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

4.10.2 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 or 1
<*channel number*>: 0 to 4 and 6 to 10

bool R_PG_Timer_StartCount_MTU_U<*unit number*>_C<*channel number*>_<*phase*> (void)
<*unit number*>: 0 or 1
<*channel number*>: 5 or 11
<*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 or 1
<*channel number*>: 0 to 11

RPDL function

R_MTU2_ControlChannel

Details

- Starts the MTU count operation.
- Call R_PG_Timer_MTU_U<*unit number*>_C<*channel number*> to make the initial settings before calling this function.

Example

A case where the setting is made as follows.

- MTU unit 0 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
}
```

4.10.3 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 or 1
<*channel number*>: 0 to 4 and 6 to 10
bool R_PG_Timer_HaltCount_MTU_U<*unit number*>_C<*channel number*>_<*phase*> (void)
<*unit number*>: 0 or 1
<*channel number*>: 5 or 11
<*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 or 1
<*channel number*>: 0 to 11

RPDL function R_MTU2_ControlChannel

Details
- Halts the MTU count operation.
- To make the MTU resume counting, call the following function.
  For MTU0 to MTU4 and MTU6 to MTU10
  R_PG_Timer_StartCount_MTU_U<*unit number*>_C<*channel number*>
  For MTU5 and MTU11
  R_PG_Timer_StartCount_MTU_U<*unit number*>_C<*channel number*>_<*phase*>

Example A case where the setting is made as follows.
- MTU unit 0 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
}
```

4.10.4 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 or 1
<*channel number*>: 0 to 4 and 6 to 10

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 or 1
<*channel number*>: 5 or 11

Description

Acquire the MTU counter value

Parameter

For MTU0 to MTU4 and MTU6 to MTU10

uint16_t * counter_val	Destination for storage of the counter value

For MTU5 and MTU11

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 or 1
<*channel number*>: 0 to 11

RPDL function

R_MTU2_ReadChannel

Details

- Acquires the counter value of a MTU.

Example

A case where the setting is made as follows.

- MTU unit 0 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 );
}
```

4.10.5 R_PG_Timer_SetCounterValue_MTU_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_SetCounterValue_MTU_U<*unit number*>_C<*channel number*>
(uint16_t counter_val)
<*unit number*>: 0 or 1 <*channel number*>: 0 to 4 and 6 to 10

bool R_PG_Timer_SetCounterValue_MTU_U<*unit number*>_C<*channel number*>_<*phase*>
(uint16_t counter_val)
<*unit number*>: 0 or 1 <*channel number*>: 5 or 11 <*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 or 1 <*channel number*>: 5 or 11

Description

Set the MTU counter value

Parameter

For MTU0 to MTU11

uint16_t counter_val	Value to be written to the counter

For MTU5 and MTU11

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

Return value

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

File for output

R_PG_Timer_MTU_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0 or 1
<*channel number*>: 0 to 11

RPDL function

R_MTU2_ControlChannel

Details

- Set the counter value of a MTU.

Example

A case where the setting is made as follows.

- MTU unit 0 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
}
```

4.10.6 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 or 1
<*channel number*>: 0 to 3or 6 to 9

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);
<*unit number*>: 0 or 1
<*channel number*>: 4 or 10

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 or 1
<*channel number*>: 5 or 11

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

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 or 1
<*channel number*>: 0 to 11

RPDL function

R_MTU2_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.
- The flags of compare match/imput capture C and D are available in channel 0, 3, 4, 6, 9 and 10.
  Specify 0 for other channels.
- The underflow flag is available in channel 1, 2, 7, and 8. Specify 0 for other channels.
- The flags of compare match E and F are available in channel 1 and 6. Specify 0 for other channels.

Example A case where the setting is made as follows.

- MTU unit 0 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

}
```

4.10.7 R_PG_Timer_StopModule_MTU_U<*unit number*>

Definition: bool R_PG_Timer_StopModule_MTU_U<*unit number*> (void)
<*unit number*>: 0 or 1

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 or 1

RPDL function: R_MTU2_Destroy

Details:

- Stops a MTU unit and places it in the module-stop state per unit. If two or more channels are running when this function is called, all channels are stopped. Call the following function to stop a single channel.
  For MTU0 to MTU4 and MTU6 to MTU10
  R_PG_Timer_HaltCount_MTU_U<*unit number*>_C<*channel number*>
  For MTU5 and MTU11
  R_PG_Timer_HaltCount_MTU_U<*unit number*>_C<*channel number*>_<*phase*>

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

- MTU unit 0 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();
}
```

4.10.8 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 or 1
<*channel number*>: 0 to 4 or 6 to 10

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 or 1
<*channel number*>: 5 or 11

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_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

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 or 1
<*channel number*>: 0 to 11

RPDL function

R_MTU2_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.
- The general register C and D are available in channel 0, 3, 6, and 9. Specify 0 for other
- channels.
  The general register E and F are available in channel 1 and 6. Specify 0 for other channels.

Example

A case where the setting is made as follows.

- MTU unit 0 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
    );
}
```

4.10.9 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*>: A, B, C, D, E, or F (for MTU0 and MTU6)
A, B, C, or D (for MTU3, MTU4, MTU9 and MTU10)
A or B (for MTU1, MTU2, MTU7 and MTU8)
U, V or W (for MTU5 and MTU11)
<*unit number*>: 0 or 1
<*channel number*>: 0 to 11

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 or 1
<*channel number*>: 0 to 11

RPDL function R_MTU2_ControlChannel

Details
- This function sets the general register value.

Example

A case where the setting is made as follows.
- MTU unit 0 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
}
```

4.10.10 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 or 1
<*channel number*>: 4 or 10

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 register's 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 or 1
<*channel number*>: 4 or 10

RPDL function R_MTU2_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 );
}
```

4.11 Port Output Enable 2 (POE2)

4.11.1 R_PG_POE_Set

Definition bool R_PG_POE_Set (void)

Description Set up the POE

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_POE.c

RPDL function R_POE_Set, R_POE_Create

- Sets up the output control of MTU0, 3, 4, 6, 9, or 10 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 this chapter end, Notes on Notification Functions.

Example

A case where the setting is made as follows.

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

```
//Include “R_PG_<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
}
```

4.11.2 R_PG_POE_SetHiZ_MTU<MTU channel number>

Definition bool R_PG_POE_SetHiZ_MTU<*MTU channel number*>(void)

<*MTU channel number*>: 0, 3_4, 6, 9_10

Description Place the MTU 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, 9, or 10 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
}
```

4.11.3 R_PG_POE_GetRequestFlagHiZ_MTU<*MTU channel number*>

Definition bool R_PG_POE_GetRequestFlagHiZ_MTU0 (bool* poe8)
bool R_PG_POE_GetRequestFlagHiZ_MTU3_4 (bool* poe0, bool* poe1, bool* poe2, bool* poe3)
bool R_PG_POE_GetRequestFlagHiZ_MTU6 (bool* poe9)
bool R_PG_POE_GetRequestFlagHiZ_MTU9_10 (bool* poe4, bool* poe5, bool* poe6, bool* poe7)

Description Acquire the high-impedance request flags

Parameter

bool* poen (n:0 to 9)	The address of storage area for POEn#(n:0 to 9) high-impedance request flags

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output R_PG_POE.c

RPDL function R_POE_GetStatus

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

Example A case where the setting is made as follows.
- MTU3 and 4 pin output has been set (Setting of MTU)
- MTU3 and 4 output pins have been set to be controlled by the high impedance request
- POE0 has been selected as a high-impedance request signal input

```
//Include “R_PG_<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_MTU3_4( &poe0, 0, 0, 0 );
    }while( ! poe0 );

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

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

4.11.4 R_PG_POE_GetShortFlag_MTU<*MTU channel number*>

Definition bool R_PG_POE_GetShortFlag_MTU3_4 (bool * detected)
bool R_PG_POE_GetShortFlag_MTU9_10 (bool * detected)

Description Acquire the MTU output short flags

Parameter

bool* detected	The address of storage area for the output short flag (MTU3,4:OSF1 or MTU9,10:OSF2)

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output R_PG_POE.c

RPDL function R_POE_GetStatus

Details
- Acquires the MTU3 ,4 or MTU9,10 complementary PWM output short flags (MTU3,4:OSF1 or MTU9,10:OSF2).

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

```
//Include “R_PG_<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_MTU3_4 (&detected);

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

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

4.11.5 R_PG_POE_ClearFlag_MTU<*MTU channel number*>

Definition bool R_PG_POE_ClearFlag_MTU<*MTU channel number*>(void)

< *MTU channel number*>: 0, 3_4, 6, 9_10

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.

MTU	Flags
0	POE8 request flag (POE8F)
3, 4	POE0 to 3 request flag (POE0F to POE3F), MTU3,4 output short flag(OSF1)
6	POE9 request flag (POE9F)
9, 10	POE4 to 7 request flag (POE4F to POE7F), MTU9,10 output short flag(OSF2)

Example A case where the setting is made as follows.

- The output enable interrupt 1 (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_MTU3_4 (&detected);

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

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

4.12 Programmable Pulse Generator (PPG)

4.12.1 R_PG_PPG_StartOutput_U<*unit number*>_G<*group number*>

Definition: bool R_PG_PPG_StartOutput_U<*unit number*>_G<*group number*> (void)
<*unit number*>: 0 or 1
<*group number*>: 0 to 7

Description: Set up the PPG and start outputting

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_PPG_U<*unit number*>.c
<*unit number*>: 0 and 1

RPDL function: R_PPG_Create

Details:
- Releases the PPG from the module-stop, makes initial settings, and starts the outputting signals from the selected pins on GUI.
- This function sets initial output value and 2nd output value.
- The MTU is not set up in this function. Use MTU function to set up the MTU.

Example: A case where the setting is made as follows.
- Pulse output pins PO8 to PO11 on group 2 have been enabled.
- MTU comparematch A interrupt has been enabled and Mtu0IcCmAIntFunc is specified as a interrupt notification function.

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

uint8_t output_val;

void func(void)
{
    output_val=1;

    R_PG_Timer_Set_MTU_U0_C0();       //Set up MTU0
    R_PG_PPG_StartOutput_U0_G2();     //Set up PPG and start output
    R_PG_Timer_StartCount_MTU_U0_C0();      // Start MTU0 count operation
}

//MTU0 compare match A interrupt notification function
void Mtu0IcCmAIntFunc (void)
{
    //Set next output value
    R_PG_PPG_SetOutputValue_U0_G2( output_val );
}
```

4.12.2 R_PG_PPG_StopOutput_U<*unit number*>_G<*group number*>

Definition

bool R_PG_PPG_StopOutput_U<*unit number*>_G<*group number*> (void)

<*unit number*>: 0 or 1

<*group number*>: 0 to 7

Description

Stop outputting

Parameter

None

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_PPG_U<*unit number*>.c

<*unit number*>: 0 and 1

RPDL function

R_PPG_Destroy

Details

- Stops the PPG output.
- If all the outputs in a unit become disabled, that unit will be put into the stop state to reduce power consumption.

Example

A case where the setting is made as follows.

- Pulse output pins PO8 to PO11 on group 2 have been enabled.
- MTU comparematch A interrupt has been enabled and Mtu0IcCmAIntFunc is specified as a interrupt notification function.

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

uint8_t output_val;

void func(void)
{
    output_val=1;

    R_PG_Timer_Set_MTU_U0_C0();       //Set up MTU0
    R_PG_PPG_StartOutput_U0_G2();     //Set up PPG and start output
    R_PG_Timer_StartCount_MTU_U0_C0();      // Start MTU0 count operation
}

//MTU0 compare match A interrupt notification function
void Mtu0IcCmAIntFunc (void)
{
    output_val++;       //Increment the output value

    R_PG_PPG_SetOutputValue_U0_G2( output_val );      //Set the next output value

    if(output_val >= 0x0f){
        R_PG_PPG_StopOutput_U0_G2();      //Stop the output
    }
}
```

4.12.3 R_PG_PPG_SetOutputValue_U<*unit number*>_G<*group number*>

Definition

bool R_PG_PPG_SetOutputValue_ U<*unit number*>_G<*group number*> (uint8_t data)

<*unit number*>: 0 or 1
<*group number*>: 0 to 7

Description

Set the output value of single group

Parameter

uint8_t output_val	Output vale an the next update (MTU compare match) Group 0,2,4,6 : bit3 to bit0 are avairable Group 1,3,5,7 : bit7 to bit4 are avairable

Parameter

None

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_PPG_U<*unit number*>.c <*unit number*>: 0 and 1

RPDL function

R_PPG_Destroy

Details

- Sets the output value of single group for next update timing (MTU compare match).
- The data is using the format:

  For group 1,3,5, and 7, set the value in upper 4 bits.

	Group 1, 3, 5 or 7				Group 0, 2, 4 or 6			
Group pair	b7	b6	b5	b4	b3	b2	b1	b0
1 & 0	PO7	PO6	PO5	PO4	PO3	PO2	PO1	PO0
3 & 2	PO15	PO14	PO13	PO12	PO11	PO10	PO9	PO8
5 & 4	PO23	PO22	PO21	PO20	PO19	PO18	PO17	PO16
7 & 6	PO31	PO30	PO29	PO28	PO27	PO26	PO25	PO24

Example

A case where the setting is made as follows.

- Pulse output pins PO4 to PO7 on group 1 have been enabled.
- MTU comparematch A interrupt has been enabled and Mtu0IcCmAIntFunc is specified as a interrupt notification function.

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

uint8_t output_val;

void func(void)
{
      output_val=1;
      R_PG_Timer_Set_MTU_U0_C0();       //Set up MTU0
      R_PG_PPG_StartOutput_U0_G2();       //Set up PPG and start output
      R_PG_Timer_StartCount_MTU_U0_C0();       // Start MTU0 count operation
}

//MTU0 compare match A interrupt notification function
void Mtu0IcCmAIntFunc (void)
{
      output_val++;      //Increment the output value
      R_PG_PPG_SetOutputValue_U0_G1( output_val << 4 ); //Set the next output value
      if(output_val >= 0x0f){
            R_PG_PPG_StopOutput_U0_G1();       //Stop the output
      }
}
```

4.12.4 R_PG_PPG_SetOutputValue_U<*unit number*>_G<*group number1*>_G<*group number2*>

Definition bool R_PG_PPG_SetOutputValue_ U<*unit number*>_G<*group number1*>_G<*group number2*>
(uint8_t data)
<*unit number*>: 0 or 1
<*group number1*>: 1, 3, 5, 7
<*group number2*>: 0, 2, 4, 6

Description Set the output value for a pair of groups

Conditions for output Pair of groups have been set

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_PPG_U<*unit number*>.c
<*unit number*>: 0 and 1

RPDL function R_PPG_Destroy

Details

- Sets the output value of a pair of groups (0-1, 2-3, 4-5, or 6-7) for next update timing (MTU compare match).
- The data is using the format:

	Group 1, 3, 5 or 7				Group 0, 2, 4 or 6			
Group pair	b7	b6	b5	b4	b3	b2	b1	b0
1 & 0	PO7	PO6	PO5	PO4	PO3	PO2	PO1	PO0
3 & 2	PO15	PO14	PO13	PO12	PO11	PO10	PO9	PO8
5 & 4	PO23	PO22	PO21	PO20	PO19	PO18	PO17	PO16
7 & 6	PO31	PO30	PO29	PO28	PO27	PO26	PO25	PO24

Example A case where the setting is made as follows.

- Pulse output groups 0 and 1 have been enabled.
- MTU comparematch A interrupt has been enabled and Mtu0IcCmAIntFunc is specified as a interrupt notification function.

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

uint8_t output_val;

void func(void)
{
      output_val=1;
      R_PG_Timer_Set_MTU_U0_C0();       //Set up MTU0
      R_PG_PPG_StartOutput_U0_G0();      //Set up PPG and start output from group 0
      R_PG_PPG_StartOutput_U0_G1();      //Set up PPG and start output from froup 1
      R_PG_Timer_StartCount_MTU_U0_C0();       // Start MTU0 count operation
}

void Mtu0IcCmAIntFunc (void)
{
      R_PG_PPG_SetOutputValue_U0_G0_G1( output_val ); //Set the next output value
}
```

4.13 8-Bit Timer (TMR)

4.13.1 R_PG_Timer_Start_TMR_U<*unit number*>(_C<*channel number*>)

Definition: bool R_PG_Timer_Start_TMR_U<*unit number*>(_C<*channel number*>) (void)
<*unit number*>: 0 or 1
<*channel number*>: 0 to 3
((_C<*channel number*>) is added in the 8-bit mode)

Description: Set up the TMR and start the count operation

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Timer_TMR_U<*unit number*>.c
<*unit number*>: 0 and 1

RPDL function: R_TMR_CreateChannel (8-bit mode)
R_TMR_CreateUnit (16-bit mode)

Details:

- Releases the TMR from the module-stop, makes initial settings, and starts the TMR counting. The initial settings are made per channel in the 8-bit mode and per unit in the 16-bit mode (when the two channels of a unit are cascade-connected).
- Interrupts of the TMR 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.
  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_TMR_U<*unit number*>(_C<*channel number*>).
- When counting driven by an externally input clock, the external reset signal, or pulse output is in use, the direction (input or output) and input buffer for the pin to be used is set in this function.

Example1

The 16-bit timer mode has been specified for TMR unit 1.

In this case, the following interrupt notification functions have been set in the GUI.

Overflow interrupt: TmrOf2IntFunc

Compare match A interrupt: TmrCma2IntFunc

Compare match B interrupt: TmrCma2IntFunc

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

void func(void)
{
    //Place TMR unit 1 in the 16-bit mode.
    R_PG_Timer_Start_TMR_U0();
}

void TmrOf2IntFunc(void)
{
    func_of();      //Processing in response to an overflow interrupt
}

void TmrCma2IntFunc(void)
{
    func_cma();     //Processing in response to a compare match A interrupt
}

void TmrCma2IntFunc(void)
{
    func_cmb();     //Processing in response to a compare match B interrupt
}
```

Example2

The 8-bit timer mode has been specified for TMR0 in the GUI.

Whether an interrupt has been requested or not is confirmed by checking the interrupt flag in the GUI.

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

void func1(void)
{
    bool cma_flag;

    //Place TMR0 in the 8-bit mode and start it counting.
    R_PG_Timer_Start_TMR_U0_C0();

    While(1){
        bool flag;
        //Acquire the compare match A interrupt request flag.
        R_PG_PG_Timer_GetRequestFlag_TMR_U0_C0( &cma_flag, 0, 0 );

        if( cma_flag ){
            func_cma0();    //Processing of IRQ0
        }
    }
}
```

4.13.2 R_PG_Timer_HaltCount_TMR_U<*unit number*>(_C<*channel number*>)

Definition

bool R_PG_Timer_HaltCount_TMR_U<*unit number*>(_C<*channel number*>) (void)

<*unit number*>: 0 or 1

<*channel number*>: 0 to 3

((_C<*channel number*>) is added in the 8-bit mode.)

Description

Halt the TMR count operation

Parameter

None

Return value

true	Halting succeeded.
false	Halting failed.

File for output

R_PG_Timer_TMR_U<*unit number*>.c

<*unit number*>: 0 or 1

RPDL function

R_TMR_ControlChannel (8-bit mode)

R_TMR_ControlUnit (16-bit mode)

Details

- Halts the TMR count operation.
- To resume the count operation, call the following function.
  R_PG_Timer_ResumeCount_TMR_U<*unit number*>(_C<*channel number*>)

Example

The 8-bit timer mode was specified for TMR0 in the GUI.
TmrCma0IntFunc was specified as the name of the compare match A interrupt function in the GUI.

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

void func(void)
{
    //Place TMR0 in the 8-bit mode.
    R_PG_Timer_Start_TMR_U0_C0();
}

void TmrCma0IntFunc(void)
{
    //Halt counting by TMR0.
    R_PG_Timer_HaltCount_TMR_U0_C0();

    func_cma();     //Processing in response to a compare match A interrupt

    //Resume counting by TMR0.
    R_PG_Timer_ResumeCount_TMR_U0_C0();
}
```

4.13.3 R_PG_Timer_ResumeCount_TMR_U<*unit number*>(_C<*channel number*>)

Definition

bool R_PG_Timer_ResumeCount_TMR_U<*unit number*>(_C<*channel number*>) (void)

<*unit number*>: 0 or 1

<*channel number*>: 0 to 3

((_C<*channel number*>) is added in the 8-bit mode.)

Description

Resume the TMR count operation

Parameter

None

Return value

true	Resuming count succeeded.
false	Resuming count failed.

File for output

R_PG_Timer_TMR_U<*unit number*>.c

<*unit number*>: 0 or 1

RPDL function

R_TMR_ControlChannel (8-bit mode)

R_TMR_ControlUnit (16-bit mode)

Details

- Resumes the TMR count operation that was halted by R_PG_Timer_HaltCount_TMR_U<*unit number*>(_C<*channel number*>).

Example

The 8-bit timer mode was selected for TMR0 in the GUI.

TmrCma0IntFunc was specified as the name of the compare match A interrupt function in the GUI.

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

void func(void)
{
      //Place TMR0 in the 8-bit mode.
      R_PG_Timer_Start_TMR_U0_C0();
}

void TmrCma0IntFunc(void)
{
      //Halt counting by TMR0.
      R_PG_Timer_HaltCount_TMR_U0_C0();

      func_cma();       //Processing in response to a compare match A interrupt

      //Resume counting by TMR0.
      R_PG_Timer_ResumeCount_TMR_U0_C0();
}
```

4.13.4 R_PG_Timer_GetCounterValue_TMR_U<*unit number*>(_C<*channel number*>)

Definition

•8-bit mode

bool R_PG_Timer_GetCounterValue_TMR_U<*unit number*>_C<*channel number*>
(uint8_t * data)

<*unit number*>: 0 or 1

<*channel number*>: 0 to 3

•16-bit mode

bool R_PG_Timer_GetCounterValue_TMR_U<*unit number*> (uint16_t * data)

<*unit number*>: 0 or 1

Description

Acquire the TMR counter value

Parameter

uint8_t * data (8-bit mode) uint16_t * data (16-bit mode)	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_TMR_U<*unit number*>.c

<*unit number*>: 0 or 1

RPDL function

R_TMR_ReadChannel (8-bit mode)

R_TMR_ReadUnit (16-bit mode)

Details

- Acquires the counter value of a TMR.
  The value of the 8-bit counter for the specified channel is stored if the TMR unit is in the 8-bit timer mode. The counter values for both channels are stored as follows if the TMR unit is in the 16-bit mode.

Unit	b15 to b8	b7 to b0
0	TMR0 counter	TMR1 counter
1	TMR2 counter	TMR3 counter

*When the TMR unit is in the 16-bit mode, the higher-order bits are in TMR0 (or TMR2).

Example

The 8-bit timer mode was selected for TMR0 in the GUI.

```
#include "R_PG_default.h" //Include "R_PG_<PDG project name>.h" to use this function.
void func1(void)
{
    R_PG_Timer_Start_TMR_U0_C0();    //Place TMR0 in the 8-bit mode.
}
uint8_t func2(void)
{
    uint8_t counter_val;
    //Acquire the value of a counter of TMR0.
    R_PG_Timer_GetCounterValue_TMR_U0_C0( &counter_val );
    return data;
}
```

4.13.5 R_PG_Timer_SetCounterValue_TMR_U<*unit number*>(_C<*channel number*>)

Definition

•8-bit mode

bool R_PG_Timer_SetCounterValue_TMR_U<*unit number*>_C<*channel number*>
(uint8_t data)

<*unit number*>: 0 or 1

<*channel number*>: 0 to 3

•16-bit mode

bool R_PG_Timer_SetCounterValue_TMR_U<*unit number*> (uint16_t data)

<*unit number*>: 0 or 1

Description

Set the TMR counter value

Parameter

uint8_t data (8-bit mode) uint16_t data (16-bit mode)	Value to be written to the counter

Return value

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

File for output

R_PG_Timer_TMR_U<*unit number*>.c

<*unit number*>: 0 or 1

RPDL function

R_TMR_ControlChannel (8-bit mode)

R_TMR_ControlUnit (16-bit mode)

Details

- Set the counter value of a TMR.
  The value of the 8-bit counter for the specified channel is stored if the TMR unit is in the 8-bit timer mode. The counter values for both channels are stored as follows if the TMR unit is in the 16-bit mode.

Unit	b15 to b8	b7 to b0
0	TMR0 counter	TMR1 counter
1	TMR2 counter	TMR3 counter

*When the TMR unit is in the 16-bit mode, the higher-order bits are in TMR0 (or TMR2).

Example

The 8-bit timer mode was selected for TMR0 in the GUI.

```
#include "R_PG_default.h" //Include "R_PG_<PDG project name>.h" to use this function.
void func1(void)
{
      //Place TMR0 in the 8-bit mode.
      R_PG_Timer_Start_TMR_U0_C0();
}
void func2(void)
{
      //Set the value of a counter of TMR0.
      R_PG_Timer_SetCounterValue_TMR_U0_C0( 0 );
}
```

4.13.6 R_PG_Timer_GetRequestFlag_TMR_U<*unit number*>(_C<*channel number*>)

Definition

bool R_PG_Timer_GetRequestFlag_TMR_U<*unit number*>_C<*channel number*>
(bool* cma, bool* cmb, bool* ov);
<*unit number*>: 0 or 1
<*channel number*>: 0 to 3
((_C<*channel number*>) is added in the 8-bit mode.)

Description

Acquire and clear the TMR interrupt flags

Parameter

bool* cma	The address of storage area for the compare match A flag
bool* cmb	The address of storage area for the compare match B 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_Timer_TMR_U<*unit number*>.c
<*unit number*>: 0 or 1

RPDL function

R_TMR_ReadChannel (8-bit mode)
R_TMR_ReadUnit (16-bit mode)

Details

- This function acquires the interrupt flags of TMR.
- 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

The 8-bit timer mode was selected for TMR0 in the GUI.

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

uint16_t counter;

void func(void)
{
    //Place TMR0 in the 8-bit mode.
    R_PG_Timer_Start_TMR_U0_C0();

    //Wait for the compare match A
    do{
        R_PG_Timer_GetRequestFlag_TMR_U0_C0(
            & cma_flag,
            0,
            0
        );
    } while( !cma_flag );

    func_cmA();     //Processing in response to a compare match A interrupt
}
```

4.13.7 R_PG_Timer_StopModule_TMR_U<*unit number*>

Definition: bool R_PG_Timer_StopModule_TMR_U<*unit number*> (void)
<*unit number*>: 0 or 1

Description: Shut down a TMR unit

Parameter: None

Return value:

true	Shutting down succeeded
false	Shutting down failed

File for output: R_PG_Timer_TMR_U<*unit number*>.c
<*unit number*>: 0 or 1

RPDL function: R_TMR_Destroy

Details:

- Stops a TMR unit and places it in the module-stop state per unit. If both TMR0 and TMR1 of unit 0 (or both TMR2 and TMR3 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_TMR_U<*unit number*>_C<*channel number*>

Example: The 8-bit timer mode was selected for TMR0 in the GUI.
TmrCma0IntFunc was specified as the name of the compare match A interrupt function in the GUI.

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

void func(void)
{
    //Place TMR0 in the 8-bit mode.
    R_PG_Timer_Start_TMR_U0_C0();
}

void TmrCma0IntFunc(void)
{
    func_cma();     //Processing in response to a compare match A interrupt

    //Stop TMR unit 0.
    R_PG_Timer_StopModule_TMR_U0();
}
```

4.14 Compare Match Timer (CMT)

4.14.1 R_PG_Timer_Start_CMT_U<*unit number*>_C<*channel number*>

Definition: bool R_PG_Timer_Start_CMT_U<*unit number*>_C<*channel number*> (void)
<*unit number*>: 0 or 1
<*channel number*>: 0 to 3

Description: Set up the CMT and start the count operation

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Timer_CMT_U<*unit number*>.c
<*unit number*>: 0 and 1

RPDL function: R_CMT_Create

Details:

- Releases the CMT from the module-stop, makes initial settings, and starts the CMT counting.
- Interrupts of the CMT are set by this function. When the name of the interrupt notification function has been specified in the GUI, if an interrupt occurs in the CPU, the function having the specified name will be called. Create the interrupt notification function as follows:
  void <*name of the interrupt notification function*> (void)
  For the interrupt notification function, note the contents of 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_Timer_Start_CMT_U0_C0 (); //Set up the CMT0 and start the count operation
}

void Cmt0IntFunc (void)
{
      func_cmt0();       //Processing in response to a compare match interrupt
}
```

4.14.2 R_PG_Timer_HaltCount_CMT<*unit number*>_C<*channel number*>

Definition: bool R_PG_Timer_HaltCount_CMT_U<*unit number*>_C<*channel number*> (void)
<*unit number*>: 0 or 1
<*channel number*>: 0 to 3

Description: Halt the CMT count operation

Parameter: None

Return value:

true	Halting succeeded.
false	Halting failed.

File for output: R_PG_Timer_CMT_U<*unit number*>.c
<*unit number*>: 0 or 1

RPDL function: R_CMT_Control

Details:
- Halts the CMT count operation.
- To resume the count operation, call the following function.
  R_PG_Timer_ResumeCount_CMT_U<*unit number*>_C<*channel number*>

Example: A case where the setting is made as follows.
- CMT unit 0 channel 0 was set up
- Cmt0IntFunc was specified as the compare match interrupt notification function name

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

void func(void)
{
    R_PG_Timer_Start_CMT_U0_C0();    //Set up the CMT0 and start the count operation
}

void Cmt0IntFunc(void)
{
    //Halt the CMT0 count operation
    R_PG_Timer_HaltCount_CMT_U0_C0();

    func_cmt0();    //Processing in response to a compare match interrupt

    //Resume the CMT0 count operation
    R_PG_Timer_ResumeCount_CMT_U0_C0();
}
```

4.14.3 R_PG_Timer_ResumeCount_CMT_U<*unit number*>_C<*channel number*>

Definition: bool R_PG_Timer_ResumeCount_CMT_U<*unit number*>_C<*channel number*> (void)
<*unit number*>: 0 or 1
<*channel number*>: 0 to 3

Description: Resume the CMT count operation

Parameter: None

Return value:

true	Resuming count succeeded.
false	Resuming count failed.

File for output: R_PG_Timer_CMT_U<*unit number*>.c
<*unit number*>: 0 or 1

RPDL function: R_CMT_Control

Details:
- Resumes the CMT count operation that was halted by R_PG_Timer_HaltCount_CMT_U<*unit number*>_C<*channel number*>.

Example: A case where the setting is made as follows.
- CMT unit 0 channel 0 was set up
- Cmt0IntFunc was specified as the compare match interrupt notification function name

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

void func(void)
{
    R_PG_Timer_Start_CMT_U0_C0();    //Set up the CMT0 and start the count operation
}

void Cmt0IntFunc(void)
{
    //Halt the CMT0 count operation
    R_PG_Timer_HaltCount_CMT_U0_C0();

    func_cmt0();    //Processing in response to a compare match interrupt

    //Resume the CMT0 count operation
    R_PG_Timer_ResumeCount_CMT_U0_C0();
}
```

4.14.4 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.

void func1(void)
{
    R_PG_Timer_Start_CMT_U0_C0();    //Set up the CMT0 and start the count operation
}

unt16_t func2(void)
{
    uint16_t data;

    // Acquire the value of a CMT0 counter
    R_PG_Timer_GetCounterValue_CMT_U0_C0( &data );

    return data;
}
```

4.14.5 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 written to the counter

Return value

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

File for output

R_PG_Timer_CMT_U<*unit number*>.c
<*unit number*>: 0 or 1

RPDL function

R_CMT_Control

Details

- Set the counter value of a CMT.

Example

A case where the setting is made as follows.

- CMT unit 0 channel 0 was set up

```
#include "R_PG_default.h" //Include "R_PG_<PDG project name>.h" to use this function.
void func1(void)
{
    R_PG_Timer_Start_CMT_U0_C0();    //Set up the CMT0 and start the count operation
}
void func2(void)
{
    R_PG_Timer_SetCounterValue_CMT_U0_C0( 0 ); // Set the value of a CMT0 counter
}
```

4.14.6 R_PG_Timer_StopModule_CMT_U<*unit number*>

Definition: bool R_PG_Timer_StopModule_CMT_U<*unit number*> (void)
<*unit number*>: 0 or 1

Description: Shut down the CMT unit

Parameter: None

Return value:

true	Shutting down succeeded
false	Shutting down failed

File for output: R_PG_Timer_CMT_U<*unit number*>.c
<*unit number*>: 0 or 1

RPDL function: R_CMT_Destroy

Details:

- Stops a CMT unit and places it in the module-stop state per unit. If both CMT0 and CMT1 of unit 0 (or both CMT2 and CMT3 of unit 1) are running when this function is called, both channels are stopped. Call the following function to stop a single channel.
  R_PG_Timer_HaltCount_CMT_U<*unit number*>_C<*channel number*>

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

- CMT unit 0 channel 0 was set up
  Cmt0IntFunc was specified as the compare match interrupt notification function name

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

void func(void)
{
    R_PG_Timer_Start_CMT_U0_C0();    //Set up the CMT0 and start the count operation
}

void Cmt0IntFunc(void)
{
    func_cmt();        //Processing in response to a compare match interrupt

    R_PG_Timer_StopModule_CMT_U0();        // Stop the CMT unit 0
}
```

4.15 Realtime Clock (RTC)

4.15.1 R_PG_RTC_Start

Definition: bool R_PG_RTC_Start(void)

Description: Set up the RTC and start the count operation

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_RTC.c

RPDL function: R_RTC_Create

Details:

- Sets up the alarm interrupt, the periodic interrupt and the 1 Hz clock output on the RTCOUT, and starts the count operation.
- Call R_PG_Clock_Set to set up the clocks and call R_PG_Clock_Start_SUB to enable the sub-clock oscillator before calling this function.
- The current time will not be set in this function. Call R_PG_RTC_SetCurrentTime to set the current time after calling R_PG_RTC_Start.
- If the alarm time has been specified in GUI, the alarm registers shall be set in 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(); //Set up the clocks
    R_PG_Clock_Start_SUB(); //Start the sub-clock oscillator
    R_PG_RTC_Start(); //Set up the RTC and start the count operation
}
```

4.15.2 R_PG_RTC_Stop

Definition bool R_PG_RTC_Stop(void)

Description Halt the RTC count operation

Parameter None

Return value

true	Shutting down succeeded
false	Shutting down failed

File for output R_PG_RTC.c

RPDL function R_RTC_Control

Details
- Halts the RTC count operation.
- To resume the count operation, call R_PG_RTC_Restart 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(); //Set up the clocks
      R_PG_Clock_Start_SUB(); //Start the sub-clock oscillator
      R_PG_RTC_Start(); //Set up the RTC and start the count operation
}

void func2(void)
{
      R_PG_RTC_Stop ();    //Stop the count operation
}

void func3(void)
{
      R_PG_RTC_Restart();    //Re-start the count operation
}
```

4.15.3 R_PG_RTC_Restart

Definition bool R_PG_RTC_Restart(void)

Description Resume the RTC count operation

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_RTC.c

RPDL function R_RTC_Control

Details

- Resumes the RTC count operation that was halted by R_PG_RTC_Stop.

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_Clock_Start_SUB(); //Start the sub-clock oscillator
      R_PG_RTC_Start(); //Set up the RTC and start the count operation
}

void func2(void)
{
      R_PG_RTC_Stop ();    //Stop the count operation
}

void func3(void)
{
      R_PG_RTC_Restart();    //Re-start the count operation
}
```

4.15.4 R_PG_RTC_SetCurrentTime

Definition

```
bool R_PG_RTC_SetCurrentTime
 (uint8_t seconds,    uint8_t minutes,    uint8_t hours,
  uint8_t day,        uint8_t month,      uint16_t year   )
```

Description

Set the current time

Alarm interrupt is enabled

Parameter

uint8_t seconds	Seconds (BCD format) (valid from 0x00 to 0x59)
uint8_t minutes	Minutes (BCD format) (valid from 0x00 to 0x59)
uint8_t hours	Hours (BCD format) (valid from 0x00 to 0x23)
uint8_t day	Day (BCD format) (valid from 0x01 to the number of days in specified month)
uint8_t month	Month (BCD format) (valid from 0x01 to 0x12)
uint16_t year	Year (BCD format) (valid from 0x0000 to 0x9999)

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_RTC.c

RPDL function

R_RTC_Control

Details

- Sets the current time.
- The value of day of week shall be calculated from year, month and day, and set to the day-of-week counter.
- When this function is called while in the count operation, counter is stopped during the counter update.
- The parameter must be set to valid value even for the alarm that is not in use.

Example A case where the setting is made as follows.

- Alarm interrupt is enabled.
- RtcAlmIntFunc has been specified as the alarm 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_Clock_Set(); //Set up the clocks
    R_PG_Clock_Start_SUB(); //Start the sub-clock oscillator
    R_PG_RTC_Start(); //Set up the RTC and start the count operation

    R_PG_RTC_SetCurrentTime( //Set the current time(22-Nov-2000 03:44:55)
        0x55, //55 seconds
        0x44, //44 minutes
        0x03, //03 hours
        0x22, //22 days
        0x11, //November
        0x2000 //2000 years
    );

    R_PG_RTC_SetAlarmTime( //Set the alarm time(22-Nov-2000 03:45:00)
        0x00, //00 seconds
        0x45, //45 minutes
        0x03, //03 hours
        0xff, //Day of week (0xff : calculated automatically)
        0x22, //22 days
        0x11, //November
        0x2000 //2000 years
    );

    // Enable year, month, day, day of week, hour, minute andd second alarm
    R_PG_RTC_Alarm_Control(
        1, // Enable second alarm
        1, // Enable minute alarm
        1, // Enable hour alarm
        1, // Enable day of week alarm
        1, // Enable day alarm
        1, // Enable month alarm
        1 // Enable year alarm
    );
}

void RtcAlmIntFunc(void)
{
    // Processing when the alarm interrupt occurs
}
```

4.15.5 R_PG_RTC_SetAlarmTime

Definition

```
bool R_PG_RTC_SetAlarmTime
( uint8_t seconds,    uint8_t minutes,    uint8_t hours,    uint8_t day_of_week,
  uint8_t day,        uint8_t month,      uint16_t year );
```

Description Set the alarm time

Conditions for output Alarm interrupt is enabled

Parameter

uint8_t seconds	Seconds (BCD format) (valid from 0x00 to 0x59)
uint8_t minutes	Minutes (BCD format) (valid from 0x00 to 0x59)
uint8_t hours	Hours (BCD format) (valid from 0x00 to 0x23)
uint8_t day_of_week	Day of week (valid from 0x00(Sunday) to 0x06(Saturday)) Specify 0xFF for automatic calculation using the values in day,month and year.
uint8_t day	Day (BCD format) (valid from 0x01 to the number of days in specified month)
uint8_t month	Month (BCD format) (valid from 0x01 to 0x12)
uint16_t year	Year (BCD format) (valid from 0x0000 to 0x9999)

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_RTC.c

RPDL function R_RTC_Control

Details
- Sets the alarm time.
- The parameter must be set to valid value even for the alarm that is not in use.

Example Refer to the example of R_PG_RTC_SetCurrentTime.

4.15.6 R_PG_RTC_Alarm_Control

Definition

```
bool R_PG_RTC_Alarm_Control
( bool sec_enable,  bool min_enable,  bool hour_enable,  bool day_of_week_enable,
  bool day_enable,  bool month_enable,  bool year_enable  );
```

Description: Enable/Disable the alarm

Conditions for output: Alarm interrupt is enabled

Parameter

bool sec_enable	Second alarm setting (1:enable 0:disable)
bool min_enable	Minute alarm setting (1:enable 0:disable)
bool hour_enable	Hour alarm setting (1:enable 0:disable)
bool day_of_week_enable	Day of week alarm setting (1:enable 0:disable)
bool day_enable	Day alarm setting (1:enable 0:disable)
bool month_enable	Month alarm setting (1:enable 0:disable)
bool year_enable	Year alarm setting (1:enable 0:disable)

Return value

true	Setting was made correctly
false	Setting failed

File for output: R_PG_RTC.c

RPDL function: R_RTC_Control

Details

- Enables or disables second, minute, hour, day of week, day, month, and year alarm.

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

- Alarm interrupt is enabled.
- RtcAlmIntFunc has been specified as the alarm 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_Clock_Set(); //Set up the clocks
    R_PG_Clock_Start_SUB(); //Start the sub-clock oscillator
    R_PG_RTC_Start(); //Set up the RTC and start the count operation
    R_PG_RTC_SetCurrentTime( //Set the current time(01-Jan-0000 22:33:44)
        0x44, //44 seconds
        0x33, //33 minutes
        0x22, //22 hours
        0x01, //01 days
        0x01, //January
        0x0000 //0000 year
    );

    R_PG_RTC_SetAlarmTime( //Set the alarm time(01-Jan-0000 22:33:55)
        0x55, //55 seconds
        0x33, //33 minutes
        0x22, //22 hours
        0xff, //Day of week (0xff : calculated automatically)
        0x01, //01 days
        0x01, //January
        0x0000 //0000 year
    );

    R_PG_RTC_Alarm_Control( // Enable hour, minute, and second alarm
        1, // Enable second alarm
        1, // Enable minute alarm
        1, // Enable hour alarm
        0, // Disable day of week alarm
        0, // Disable day alarm
        0, // Disable month alarm
        0 // Disable year alarm
    );
}

void RtcAlmIntFunc(void)
{
    // Processing when the alarm interrupt occurs
}
```

4.15.7 R_PG_RTC_Adjust30sec

Definition: bool R_PG_RTC_Adjust30sec(void)

Description: Start the 30-second adjustment process

Conditions for output: Alarm interrupt is enabled

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_RTC.c

RPDL function: R_RTC_Control

Details:

- Starts 30-second adjustment (30 seconds or less are rounded down to 00 second, and 30 seconds or more are rounded up to one minute) process.

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_Clock_Start_SUB(); //Start the sub-clock oscillator
    R_PG_RTC_Start(); //Set up the RTC and start the count operation
}

void func2(void)
{
    R_PG_RTC_Adjust30sec(); //Start the 30-second adjustment process
}
```

4.15.8 R_PG_RTC_SetPeriodicInterrupt

Definition: bool R_PG_RTC_SetPeriodicInterrupt(float frequency)

Description: Set the interval for the periodic interrupt

Conditions for output: The periodic interrupt has been set

Parameter

float frequency	The frequency of the periodic interrupt request (Hz) (valid value : 0.5, 1, 2, 4, 16, 64, 256)

Return value

true	Setting was made correctly
false	Setting failed

File for output: R_PG_RTC.c

RPDL function: R_RTC_Control

Details

- Sets the interval for the periodic interrupt.

Example

A case where the setting is made as follows.

- The periodic interrupt has been set
- RtcPrdIntFunc has been specified as periodic 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_Clock_Set(); //Set up the clocks
    R_PG_Clock_Start_SUB(); //Start the sub-clock oscillator
    R_PG_RTC_Start(); //Set up the RTC and start the count operation
}

void RtcAlmIntFunc(void)
{
    // Processing when the periodic interrupt occurs

    //Set the interval of the periodic interrupt to 1/4 seconds
    R_PG_RTC_SetPeriodicInterrupt( 4 );
}
```

4.15.9 R_PG_RTC_GetStatus

Definition

```
bool R_PG_RTC_GetStatus
( uint8_t * seconds,   uint8_t * minutes,   uint8_t * hours,   uint8_t * day_of_week,
  uint8_t * day,       uint8_t * month,     uint16_t * year,    bool * carry,
  bool * alarm,        bool * period,       bool * adjustment,  bool * reset,
  bool * running   )
```

Description

Get the status of RTC

Parameter

uint8_t * seconds	The address of storage area for the current seconds
uint8_t * minutes	The address of storage area for the current minutes
uint8_t * hours	The address of storage area for the current hours
uint8_t * day_of_week	The address of storage area for the current day of the week
uint8_t * day	The address of storage area for the current day
uint8_t * month	The address of storage area for the current month
uint16_t * year	The address of storage area for the current year
bool * carry	The address of storage area for the carry interrupt flag
bool * alarm	The address of storage area for the alarm interrupt flag
bool * period	The address of storage area for the periodic interrupt flag
bool * adjustment	The address of storage area for the 30-second adjustment bit (0:Normal operation 1:Adjustment in progress)
bool * reset	The address of storage area for the reset bit (0:Normal operation 1:Reset in progress)
bool * running	The address of storage area for the start bit (0:Clock is stopped 1:Clock is running)

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output

R_PG_RTC.c

RPDL function

R_RTC_Read

Details

- Acquires the status of RTC.
- Specify the address of storage area for an item to be acquired. Specify 0 for an item that is not required.
- The interrupt flags will be cleared in this function.
- If the carry flag is read as 1, the current time and date were updated during the read process and should be re-read.

Example

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

uint8_t seconds;
uint8_t minutes;
uint8_t hours;
bool carry;

void func1(void)
{
      R_PG_Clock_Set(); //Set up the clocks
      R_PG_Clock_Start_SUB(); //Start the sub-clock oscillator
      R_PG_RTC_Start(); //Set up the RTC and start the count operation
}

void func2(void)
{
      do{
            //Get the current time and the carry flag
            bool R_PG_RTC_GetStatus(
                  &seconds,    //Seconds
                  &minutes,    //Minutes
                  &hours,      //Hours
                  0,           //Day of week
                  0,           //Day
                  0,           //Month
                  0,           //Year
                  &carry,      //Carry interrupt flag
                  0,           //Alarm interrupt flag
                  0,           //Periodic interrupt flag
                  0,           //Adjustment bit
                  0,           //Reset bit
                  0            //Start bit
            );
      } while( carry );
}
```

4.15.10 R_PG_RTC_ClockOut_Disable

Definition: bool R_PG_RTC_ClockOut_Disable (void)

Description: Enable the clock output

Conditions for output: The 1 Hz clock output on the RTCOUT has been set

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_RTC.c

RPDL function: R_RTC_Control

Details:

- Stops the 1 Hz clock output on the RTCOUT.

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

- The 1 Hz clock output on the RTCOUT has been set

```
//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();

    //Start the sub-clock oscillator
    R_PG_Clock_Start_SUB();

    //Set up the RTC, start the count operation, and start the clock output
    R_PG_RTC_Start();
}

void func2(void)
{
    R_PG_RTC_ClockOut_Disable();    //Stop the clock output
}

void func3(void)
{
    R_PG_RTC_ClockOut_Enable();    //Start the clock output
}
```

4.15.11 R_PG_RTC_ClockOut_Enable

Definition: bool R_PG_RTC_ClockOut_Enable (void)

Description: Disable the clock output

Conditions for output: The 1 Hz clock output on the RTCOUT has been set

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_RTC.c

RPDL function: R_RTC_Control

Details:

- Start the 1 Hz clock output on the RTCOUT

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

- The 1 Hz clock output on the RTCOUT has been set

```
//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();

    //Start the sub-clock oscillator
    R_PG_Clock_Start_SUB();

    //Set up the RTC, start the count operation, and start the clock output
    R_PG_RTC_Start();
}

void func2(void)
{
    R_PG_RTC_ClockOut_Disable();    //Stop the clock output
}

void func3(void)
{
    R_PG_RTC_ClockOut_Enable();    //Start the clock output
}
```

4.16 Watchdog Timer (WDT)

4.16.1 R_PG_Timer_Start_WDT

Definition: bool R_PG_Timer_Start_WDT (void)

Description: Set up the WDT and start the count operation

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Timer_ WDT.c

RPDL function: R_WDT_Create

Details:

- Makes initial settings of WDT and starts the count operation.
  When the WDT is set to interval timer mode, the interval timer interrupt is set in this function. When the name of the interrupt notification function has been specified in the GUI, if an interrupt occurs in the CPU, the function having the specified name will be called. Create the interrupt notification function as follows:
  void <*name of the interrupt notification function*> (void)
  For the interrupt notification function, note the contents of this chapter end, Notes on Notification Functions.

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

- The WDT has been set to interval timer mode.
- WdtIntFunc has been specified as a interval timer interrupt notification function name.

```
//Include “R_PG_<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
}

void WdtIntFunc(void)
{
    //Processing when the interval timer interrupt occurs
}
```

4.16.2 R_PG_Timer_HaltCount_WDT

Definition bool R_PG_Timer_HaltCount_WDT (void)

Description Stop the count operation

Parameter None

Return value

true	Halting succeeded
false	Halting failed

File for output R_PG_Timer_ WDT.c

RPDL function R_WDT_Control

Details
- Stops the WDT count operation.
- Call R_PG_Timer_Start_WDT to resume the count operation.

Example A case where the setting is made as follows.
- The WDT has been set to interval timer mode.
- WdtIntFunc has been specified as a interval timer interrupt notification function name.

```
//Include “R_PG_<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
}

void WdtIntFunc(void)
{
    R_PG_Timer_HaltCount_WDT();    //Halt the WDT count operation

    //Processing when the interval timer interrupt occurs

    R_PG_Timer_Start_WDT();    //Set up the WDT and start the count operation
}
```

4.16.3 R_PG_Timer_ResetCounter_WDT

Definition: bool R_PG_Timer_ResetCounter_WDT(void)

Description: Reset the counter of WDT

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Timer_ WDT.c

RPDL function: R_WDT_Control

Details:

- Resets the counter of WDT

Example:

```
//Include “R_PG_<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_ResetCounter_WDT();      //Reset the WDT counter
}
```

4.16.4 R_PG_Timer_ClearOverflowFlag_WDT

Definition: bool R_PG_Timer_ClearOverflowFlag_WDT (bool* ov)

Description: Reset the counter of WDT

Parameter

bool* ov	The address of storage area for the overflow flag

Return value

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Timer_ WDT.c

RPDL function: R_WDT_Control

Details

- This function acquires the counter overflow flags and clears.
- Specify 0 for a parameter if the flag is not required.

Example

A case where the setting is made as follows.

- The WDT has been set to interval timer mode.
- The priority level of interval timer interrupt has been set to 0.

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

bool ov;

void func(void)
{
    R_PG_Clock_Set();    //Set up the clocks
    R_PG_Timer_Start_WDT();    //Set up the WDT and start the count operation

    do{
        R_PG_Timer_ClearOverflowFlag_WDT( &ov ); //Get the overflow flag
    }while( !ov );

    /Processing when the interval timer interrupt occurs
}
```

4.17 Independent Watchdog Timer (IWDT)

4.17.1 R_PG_Timer_Set_IWDT

Definition: bool R_PG_Timer_Set_IWDT (void)

Description: Set up the IWDT

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_Timer_IWDT.c

RPDL function: R_IWDT_Set

Details:

- Sets up the IWDT.
- The IWDT count operation starts by counter refresh.
- R_PG_Timer_RefreshCounter_IWDT can be used to refresh the counter

Example:

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

uint8_t output_val;

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

      //Set up the IWDT
      R_PG_Timer_Set_WDT();

      //Start the count operation by refreshing the counter
      R_PG_Timer_RefreshCounter_IWDT();
}

void func2(void)
{
      //Refresh the counter
      R_PG_Timer_RefreshCounter_IWDT();
}
```

4.17.2 R_PG_Timer_RefreshCounter_IWDT

Definition: bool R_PG_Timer_RefreshCounter_IWDT (void)

Description: Refresh the counter

Parameter: None

Return value:

true	Refreshing succeeded
false	Refreshing failed

File for output: R_PG_Timer_IWDT.c

RPDL function: R_IWDT_Control

Details:

- Refreshes the IWDT counter
- To start the count operation, call this function after setting up IWDT by R_PG_Timer_Set_IWDT.
- After starting the count operation, call this function to clear the counter before the counter underflow.

Example:

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

uint8_t output_val;

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

    //Set up the IWDT
    R_PG_Timer_Set_WDT();

    //Start the count operation by refreshing the counter
    R_PG_Timer_RefreshCounter_IWDT();
}

void func2(void)
{
    //Refresh the counter
    R_PG_Timer_RefreshCounter_IWDT();
}
```

4.17.3 R_PG_Timer_GetCounterValue_IWDT

Definition: bool R_PG_Timer_GetCounterValue_IWDT(uint16_t * counter_val)

Description: Acquire the IWDT counter value

Parameter

uint16_t * counter_val	The address of storage area for the IWDT counter value

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output: R_PG_Timer_IWDT.c

RPDL function: R_IWDT_Read

Details

- Acquires the IWDT counter value.
- The underflow flag shall be cleared in this function.

Example

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

uint16_t counter_val;

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

    //Set up the IWDT
    R_PG_Timer_Set_WDT();

    //Start the count operation by refreshing the counter
    R_PG_Timer_RefreshCounter_IWDT();
}

void func2(void)
{
    R_PG_Timer_GetCounterValue_IWDT( &counter_val );

    if( counter_val < 0x1000){

        //Refresh the counter
        R_PG_Timer_RefreshCounter_IWDT();
    }
}
```

4.17.4 R_PG_Timer_ClearUnderflowFlag_IWDT

Definition: bool R_PG_Timer_ClearUnderflowFlag_IWDT(bool * un)

Description: Acquire and clear the underflow flag

Parameter

bool * un	The address of storage area for the underflow flag

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output: R_PG_Timer_IWDT.c

RPDL function: R_IWDT_Read

Details

- Acquires and clears the underflow flag.
- Specify 0 for a parameter if the flag is not required.

Example

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

bool un;

void func(void)
{
    R_PG_Timer_ClearUnderflowFlag_IWDT ( &un );
    if(un){
        // Processing after a reset caused by a couter underflow
    }
}
```

4.18 Serial Communications Interface (SCIa)

4.18.1 R_PG_SCI_Set_C<*channel number*>

Definition: bool R_PG_SCI_Set_C<*channel number*> (void)
<*channel number*>: 0 to 3, 5 to 6

Description: Set up a SCI channel

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_SCI_C<*channel number*>.c
<*channel number*>: 0 to 3, 5 to 6

RPDL function: R_SCI_Create

Details:

- Releases a SCI channel from the module-stop state, makes initial settings, and the direction (input or output) and input buffer for the pin to be used is set.
- Function R_PG_Clock_Set must be called before calling this function.
- When the name of the notification function has been specified in the GUI, if corresponding event occurs, the function having the specified name will be called. Create the notification function as follows:
  void <*name of the notification function*> (void)
  For the notification function, note the contents of this chapter end, Notes on Notification Functions.
- For pin TXD5 it is not possible for this function to ensure that external bus signals CS4 or CS7 are not output. If channel SCI5 is used for transmission, the pin TXD5 cannot be used as CS4#_D or CS7#_D.

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.
}
```

4.18.2 R_PG_SCI_StartSending_C<*channel number*>

Definition

bool R_PG_SCI_StartSending_C<*channel number*> (uint8_t * data, uint16_t count)

<*channel number*>: 0 to 3, 5 to 6

Description

Start the data transmission

Conditions for output

- The function of transmission is selected for a SCI channel in GUI.
- "Notify the transmission completion of all data by function call" is selected as the data transmission method in GUI.

Parameter

uint8_t * data	The start address of the data to be sent.
uint16_t count	The number of the data to be sent. Set this to 0 if the transmit data is a character string (ending with a null character).

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_SCI_C<*channel number*>.c

<*channel number*>: 0 to 3, 5 to 6

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 aquired 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)
{
    //Shut down the SCI0
    R_PG_SCI_StopModule_C0();
}
```

4.18.3 R_PG_SCI_SendAllData_C<*channel number*>

Definition

bool R_PG_SCI_SendAllData_C<*channel number*> (uint8_t * data, uint16_t count)

<*channel number*>: 0 to 3, 5 to 6

Description

Transmit all data

Conditions for output

- The function of transmission is selected for a SCI channel in GUI.
- Other than "Notify the transmission completion of all data by function call" is selected as the data transmission method in GUI.

Parameter

uint8_t * data	The start address of the data to be sent.
uint16_t count	The number of the data to be sent. Set this to 0 if the transmit data is a character string (ending with a null character).

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_SCI_C<*channel number*>.c

<*channel number*>: 0 to 3, 5 to 6

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.
- For usage of function for transferring the SCI transmission data by DMAC, refer to 4.4.1 R_PG_DMAC_Set_C<*channel number*>.

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
}
```

4.18.4 R_PG_SCI_GetSentDataCount_C<*channel number*>

Definition

bool R_PG_SCI_GetSentDataCount_C<*channel number*> (uint16_t * count)

<*channel number*>: 0 to 3, 5 to 6

Description

Acquire the number of transmitted data

Conditions for output

The function of transmission is selected for a SCI channel and "Notify the transmission completion of all data by function call" is selected as the data transmission method in GUI.

Parameter

uint16_t * count	The storage location for the number of bytes that have been transmitted in the current transmission.

Return value

true	Acquisition of the data count succeeded
false	Acquisition of the data count failed

File for output

R_PG_SCI_C<*channel number*>.c

<*channel number*>: 0 to 3, 5 to 6

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_Send_C0(data, 255);        //Send 255 bytes of binary data.
}

//The transmit end notification function that called when all bytes have been sent
void Sci0TrFunc(void)
{
    //Shut down the SCI0
    R_PG_SCI_StopModule_C0();
}

//The function to check the number of transmitted data and terminate the transmission
void func_terminate_SCI(void)
{
    uint16_t count;
    // Acquire the number of transmitted data
    R_PG_SCI_GetSentDataCount_C0(&count);

    if( count > 32 ){
        R_PG_SCI_StopCommunication_C0();    //Terminate the transmission
    }
}
```

4.18.5 R_PG_SCI_StartReceiving_C<*channel number*>

Definition

bool R_PG_SCI_StartReceiving_C<*channel number*> (uint8_t * data, uint16_t count)

<*channel number*>: 0 to 3, 5 to 6

Description

Start the data reception

Conditions for output

- The function of reception is selected for a SCI channel in GUI
- "Notify the reception completion of all data by function call" is selected as the data reception method in GUI

Parameter

uint8_t * data	The start address of the storage area for the expected data.
uint16_t count	The number of the data to be received.

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_SCI_C<*channel number*>.c

<*channel number*>: 0 to 3, 5 to 6

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 aquired 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)
{
    //Shut down the SCI0
    R_PG_SCI_StopModule_C0();
}
```

4.18.6 R_PG_SCI_ReceiveAllData_C<*channel number*>

Definition

bool R_PG_SCI_ReceiveAllData_C<*channel number*> (uint8_t * data, uint16_t count)

<*channel number*>: 0 to 3, 5 to 6

Description

Receive all data

Conditions for output

- The function of reception is selected for a SCI channel in GUI.
- Other than "Notify the reception completion of all data by function call" is selected as the data reception method in GUI

Parameter

uint8_t * data	The start address of the storage area for the expected data.
uint16_t count	The number of the data to be received.

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_SCI_C<*channel number*>.c

<*channel number*>: 0 to 3, 5 to 6

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.
- For usage of function for receiving the SCI transmission data by DMAC, refer to 4.4.1 R_PG_DMAC_Set_C<*channel number*>.

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
}
```

4.18.7 R_PG_SCI_StopCommunication_C<*channel number*>

Definition R_PG_SCI_StopCommunication_C<*channel number*> (void)
<*channel number*>: 0 to 3, 5 to 6

Description Stop transmission and reception of serial data

Parameter None

Return value

true	Setting was made correctly
false	Setting failed

File for output R_PG_SCI_C<*channel number*>.c
<*channel number*>: 0 to 3, 5 to 6

RPDL function R_SCI_Control

Details
- This function stops data transmission and reception.
- When "Notify the transmission completion of all data by function call" is selected as the data transmission method in GUI, the reception can be terminated by calling this function before the number of bytes specified at R_PG_SCI_StartSending_C<*channel number*> have been received.
- When "Notify the reception completion of all data by function call" is selected as the data reception method in GUI, the reception can be terminated by calling this function before the number of bytes specified at R_PG_SCI_StartReceiving_C<*channel number*> have been received.

Example
SCI0 has been set as receiver in the GUI.
Sci0ReFunc was specified as the name of the receive end notification function in the GUI.

```
//Include "R_PG_<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);          //Send 255 bytes of binary data.
}

//The receive end notification function that called when all bytes have been received.
void Sci0ReFunc(void)
{
    //Shut down the SCI0
    R_PG_SCI_StopModule_C0();
}

//The function to check the number of received data and terminate the reception
void func_terminate_SCI(void)
{
    uint8_t count;
    //Acquire the number of received data
    R_PG_SCI_GetReceivedDataCount_C0(&count);
    if( count > 32 ){
        R_PG_SCI_StopCommunication_C0();    //Terminate the reception
    }
}
```

4.18.8 R_PG_SCI_GetReceivedDataCount_C<*channel number*>

Definition

bool R_PG_SCI_GetReceivedDataCount_C<*channel number*> (uint16_t * count)

<*channel number*>: 0 to 3, 5 to 6

Description

Acquire the number of received data

Conditions for output

The function of reception is selected for a SCI channel and "Notify the reception completion of all data by function call" is selected as the data reception method in GUI.

Parameter

uint16_t * count	The storage location for the number of bytes that have been received in the current reception process.

Return value

true	Acquisition of the data count succeeded
false	Acquisition of the data count failed

File for output

R_PG_SCI_C<*channel number*>.c

<*channel number*>: 0 to 3, 5 to 6

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_Receive_C0(data, 255);         //Send 255 bytes of binary data.
}

//The receive end notification function that called when all bytes have been received.
void Sci0ReFunc(void)
{
    //Shut down the SCI0
    R_PG_SCI_StopModule_C0();
}

//The function to check the number of received data and terminate the reception
void func_terminate_SCI(void)
{
    uint16_t count;
    //Acquire the number of received data
    R_PG_SCI_GetReceivedDataCount_C0(&count);
    if( count > 32 ){
        R_PG_SCI_StopReceiving_C0();    //Terminate the reception
    }
}
```

4.18.9 R_PG_SCI_GetReceptionErrorFlag_C<*channel number*>

Definition

bool R_PG_SCI_GetReceptionErrorFlag_C<*channel number*>
(bool * parity, bool * framing, bool * overrun)
<*channel number*>: 0 to 3, 5 to 6

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 to 3, 5 to 6

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_Receive_C0(data, 1);          //Send 1bytes of binary data.
}

//The receive end notification function that called when all bytes have been received.
void Sci0ReFunc(void)
{
    // Acquire the reception error flags
    R_PG_SCI_GetReceptionErrorFlag_C0( &parity, &framing, & overrun );
}
```

4.18.10 R_PG_SCI_GetTransmitStatus_C<*channel number*>

Definition

bool R_PG_SCI_GetTransmitStatus_C<*channel number*> (bool * complete)

<*channel number*>: 0 to 3, 5 to 6

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

Return value

true	Acquisition of the transmission status succeeded
false	Acquisition of the transmission status failed

File for output

R_PG_SCI_C<*channel number*>.c

<*channel number*>: 0 to 3, 5 to 6

RPDL function

R_SCI_GetStatus

Details

- This function acquires the state of transmission.

  Transmission completion flag

0	Active
1	Complete

Example

Refer to the example 2 of R_PG_DMAC_Set_C<*channel number*>

4.18.11 R_PG_SCI_SendTargetStationID_C<*channel number*>

Definition

bool R_PG_SCI_SendTargetStationID_C<*channel number*> (uint8_t id)

<*channel number*>: 0 to 3, 5 to 6

Description

Transmits the ID code of the receiving station

Conditions for output

- The function of transmission is selected for a SCI channel
- The multi-processer communications function is enabled in the asynchronous serial communication mode

Parameter

uint8_t id	The ID to be transmitted (0 to 255)

Return value

true	Transmission succeeded
false	Transmission failed

File for output

R_PG_SCI_C<*channel number*>.c

<*channel number*>: 0 to 3, 5 to 6

RPDL function

R_SCI_Send

Details

- Generates an ID transmission cycle to transmit the ID code of the destination receiving station.
- This function waits until the ID transmission cycle has been completed.

Example

A case where the setting is made as follows.

- The function of transmission is selected for a SCI2 channel
- The multi-processer communications function is enabled in the asynchronous serial communication mode
- “Wait at the transmission function until all data has been transmitted” is selected as the data transmission method.

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

uint8_t data[10] = ABCDEFGHIJ;

void func(void)
{
    R_PG_Clock_Set();              //The clock-generation circuit has to be set first.
    R_PG_SCI_Set_C2();             //Set up SCI2
    R_PG_SCI_SendTargetStationID_C2( 5 );          //Send ID code (ID:5)
    R_PG_SCI_SendAllData_C2( data, 10 );           //Send data
}
```

4.18.12 R_PG_SCI_ReceiveStationID_C<*channel number*>

Definition

bool R_PG_SCI_ReceiveStationID_C<*channel number*> (void)

<*channel number*>: 0 to 3, 5 to 6

Description

Receives the ID code matches the ID of the receiving station itself

Conditions for output

- The function of reception is selected for a SCI channel
- The multi-processer communications function is enabled in the asynchronous serial communication mode

Parameter

None

Return value

true	Reception succeeded
false	Reception failed

File for output

R_PG_SCI_C<*channel number*>.c

<*channel number*>: 0 to 3, 5 to 6

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 SCI5 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_C5();             //Set up SCI5
    R_PG_SCI_ReceiveStationID_C5();            //Wait an ID reception
    R_PG_SCI_StartReceiving_C5( data, 10 );            //Start receiving
}
```

4.18.13 R_PG_SCI_StopModule_C<*channel number*>

Definition bool R_PG_SCI_StopModule_C<*channel number*> (void)
<*channel number*>: 0 to 3, 5 to 6

Description Shut down a SCI channel

Parameter None

Return value

true	Shutting down succeeded
false	Shutting down failed

File for output R_PG_SCI_C<*channel number*>.c
<*channel number*>: 0 to 3, 5 to 6

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 transmitter 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_Send_C0(data, 255);   //Send 255 bytes of binary data.
    R_PG_SCI_StopModule_C0();      //Shut down the SCI0
}
```

4.19 CRC Calculator (CRC)

4.19.1 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 data;

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 (&data);    // Read the CRC calculation result
    R_PG_CRC_StopModule();    // Shutdown the CRC unit
}
```

4.19.2 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.

4.19.3 R_PG_CRC_GetResult

Definition: bool R_PG_CRC_GetResult (uint16_t * data)

Description: Get the the result of calculation

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 the result of calculation

Example: Refer to the example of R_PG_CRC_Set.

4.19.4 R_PG_CRC_StopModule

Definition bool R_PG_CRC_StopModule (void)

Description Shut down CRC calculator

Parameter None

Return value

true	Shutting down succeeded
false	Shutting down failed

File for output R_PG_CRC.c

RPDL function R_CRC_Destroy

Details
- Stops the CRC calculator and places it in the module-stop state.

Example Refer to the example of R_PG_CRC_Set.

4.20 I2C Bus Interface (RIIC)

4.20.1 R_PG_I2C_Set_C<*channel number*>

Definition: bool R_PG_I2C_Set_C<*channel number*> (void)
<*channel number*>: 0 or 1

Description: Set up an I2C bus interface channel

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_I2C_C<*channel number*>.c
<*channel number*>: 0 or 1

RPDL function: R_IIC_Create

Details:

- Releases an I2C bus interface channel from the module-stop state, makes initial settings, and the direction (input or output) and input buffer for the pin to be used is set. Function R_PG_Clock_Set must be called before calling this function.

Example: RIIC0 has been set in the GUI.

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

4.20.2 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 or 1

Description

Master data reception

Conditions for output

The function of master is selected for an I2C bus interface channel in GUI.

Parameter

bool addr_10bit	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 or 1

RPDL function

R_IIC_MasterReceive

Details

- This function reads data from slave module. The stop condition is generated when the specified number of data has been received and reception completes.
- If "Wait at the reception function until all data has been transmitted" is selected as the master reception method in GUI, this function waits until the last byte has been received.
- If "Notify the reception completion of all data by function call" is selected as the master reception method in GUI, this function returns immediately and the notification function having the specified name will be called when the last byte has been received.
  Create the notification function as follows:
  void <*name of the notification function*> (void)
  For the notification function, note the contents of 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"

// The storage area for the received data
uint8_t iic_data[10];

void func(void)
{
      //The clock-generation circuit has to be set first
      R_PG_Clock_Set();

      //Set up RIIC0
      R_PG_I2C_Set_C0();

      //Master reception
     R_PG_I2C_MasterReceive_C0(
          0,          // Slave address format
          6,          // Slave address
          iic_data,   // The start address of the storage area for the received data
          10          // The number of the data to be received
     );

      //Stop RIIC0
     R_PG_I2C_StopModule_C0();
}
```

4.20.3 R_PG_I2C_MasterReceiveLast_C<*channel number*>

Definition

bool R_PG_I2C_MasterReceiveLast_C< *channel number* >
(uint8_t* data)
< *channel number* >: 0,1

Description

Complete a master reception process

Conditions for output

- The function of master is selected for an I2C bus interface channel in GUI.
- Select 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 or 1

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 RIIC0)

  Transfer system : Single-operand transfer

  Unit data size : 1 byte

  Single operand data count : 1

  Total transfer data size : Number of data 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 function name : Dmac0IntFunc

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

void Dmac0IntFunc(){
    uint8_t data; //Storage area of extra data

    //Issue NACK and STOP condition and complete the reception
    R_PG_PG_I2C_MasterReceiveLast( &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_PG_DMAC_Set_C0();

    //Activate the DMAC0
    R_PG_PG_DMAC_Activate_C0();

    //Master reception
    //For DMAC transfer, set PDL_NO_PTR for the address of the storage area
    //For DMAC transfer, set 0 for the number of the data
    R_PG_PG_I2C_MasterReceive_C0(
        0,      //Slave address format
        6,      //Slave address
        PDL_NO_PTR, // The address of the storage area
        0       // The number of the data
    );
}
```

4.20.4 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 or 1

Description

Master data transmission

Conditions for output

The function of master is selected for an I2C bus interface channel in GUI.

Parameter

bool addr_10bit	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 or 1

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();
}
```

4.20.5 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 or 1

Description

Master data transmission (No stop condition)

Conditions for output

The function of master is selected for an I2C bus interface channel in GUI.

Parameter

bool addr_10bit	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 or 1

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();
}
```

4.20.6 R_PG_I2C_GenerateStopCondition_C<*channel number*>

Definition: bool R_PG_I2C_GenerateStopCondition_C<*channel number*> (void)
<*channel number*>: 0 or 1

Description: Generate a stop condition

Conditions for output: The function of master is selected for an I2C bus interface channel in GUI.

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_I2C_C<*channel number*>.c
<*channel number*>: 0 or 1

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();
}
```

4.20.7 R_PG_I2C_GetBusState_C<*channel number*>

Definition

bool R_PG_I2C_GetBusState_C<*channel number*> (bool *busy)

<*channel number*>: 0 or 1

Description

Get the bus state

Conditions for output

The function of master is selected for an I2C bus interface channel in GUI.

Parameter

bool *busy	The address of 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 or 1

RPDL function

R_IIC_GetStatus

Details

- This function acquires the bus busy detection flag.

Bus busy detection flag

0	The I2C bus is released (bus free state)
1	The I2C bus is occupied (bus busy state or in the bus free state)

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
bool busy;
void func(void)
{
    //The clock-generation circuit has to be set first
    R_PG_Clock_Set();
    //Set up RIIC0
    R_PG_I2C_Set_C0();
    // Wait for the I2C bus to be free
    do{
        R_PG_I2C_GetBusState_C0( & busy );
    } while( busy );
    //Master transmission
    R_PG_I2C_MasterSend_C0(
        0,          // Slave address format
        6,          // Slave address
        iic_data,   // The start address of the storage area for the data to be transmitted
        10          // The number of the data to be transmitted
    );
```

4.20.8 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 or 1

Description

Slave bus monitor

Conditions for output

The function of slave is selected for an I2C bus interface channel in GUI.

Parameter

uint8_t* data	The start address of the received data
uint16_t count	The number of the data to be received

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_I2C_C<*channel number*>.c

<*channel number*>: 0 or 1

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_PG_I2C_GetTR_C0(&transmit);

    //Get a detected address
    R_PG_I2C_GetDetectedAddress_C0(&addr0, &addr1, 0, 0, 0, 0);

    if (start && transmit && addr0) {
        R_PG_I2C_SlaveSend_C(
            iic_data_tr_0,
            10
        );
    }

    else if (start && transmit && addr1) {
        R_PG_I2C_SlaveSend_C(
            iic_data_tr_1,
            10
        );
    }
}
```

4.20.9 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 or 1

Description

Slave data transmission

Conditions for output

The function of slave is selected for an I2C bus interface channel in GUI.

Parameter

uint8_t* data	The start address of the data to be transmitted
uint16_t count	The number of the data to be transmitted

Return value

true	Setting was made correctly
false	Setting failed

File for output

R_PG_I2C_C<*channel number*>.c

<*channel number*>: 0 or 1

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*>

4.20.10 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 or 1

Description

Get the detected address

Conditions for output

The function of slave is selected for an I2C bus interface channel in GUI.

Parameter

bool *addr0	The address of 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 or 1

RPDL function

R_IIC_GetStatus

Details

- This function acquires the detected address.
- Specify the address of storage area for the flags to be acquired.
- Specify 0 for a flag that is not required.
- The flag of detected address will be set to 1.

Example

Refer to the example of R_PG_I2C_SlaveMonitor_C<*channel number*>

4.20.11 R_PG_I2C_GetTR_C<*channel number*>

Definition

bool R_PG_I2C_GetTR_C<*channel number*> (bool * transmit)

<*channel number*>: 0 or 1

Description

Get the transmit/receive mode

Conditions for output

The function of slave is selected for an I2C bus interface channel in GUI.

Parameter

bool * transmit	The address of storage area for the transmit/receive mode flag Transmit/receive mode flag 0: Receive mode 1: Transmit mode

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output

R_PG_I2C_C<*channel number*>.c

<*channel number*>: 0 or 1

RPDL function

R_IIC_GetStatus

Details

- This function acquires the transmit/receive mode flag.

Example

Refer to the example of R_PG_I2C_SlaveMonitor_C<*channel number*>

4.20.12 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 or 1

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 or 1

RPDL function

R_IIC_GetStatus

Details

- This function acquires the detected event.
- Specify 0 for a flag that is not required.
- The flags of the detected event will be set to 1.

Example

Refer to the example of R_PG_I2C_SlaveMonitor_C<*channel number*>

4.20.13 R_PG_I2C_GetReceivedDataCount_C<*channel number*>

Definition

bool R_PG_I2C_GetReceivedDataCount_C<*channel number*> (uint16_t *count)

<*channel number*>: 0 or 1

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 or 1

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(
        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 );
}
```

4.20.14 R_PG_I2C_GetSentDataCount_C<*channel number*>

Definition bool R_PG_I2C_GetSentDataCount_C<*channel number*> (uint16_t *count)
<*channel number*>: 0 or 1

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 or 1

RPDL function R_IIC_GetStatus

Details
- This function acquires the number of bytes that have been transmitted in the current transmission process.

Example A case where the setting is made as follows.
- The function of master is selected for a RIIC0
- "Notify the transmission completion of all data by function call" is selected as the data transmission method

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

4.20.15 R_PG_I2C_Reset_C<*channel number*>

Definition

bool R_PG_I2C_Reset_C<*channel number*> (void)

<*channel number*>: 0 or 1

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 or 1

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(
        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();
    }
}
```

4.20.16 R_PG_I2C_StopModule_C<*channel number*>

Definition bool R_PG_I2C_StopModule_C<*channel number*> (void)
<*channel number*>: 0 or 1

Description Shut down the I2C bus interface channel

Parameter None

Return value

true	Shutting down succeeded
false	Shutting down failed

File for output R_PG_I2C_C<*channel number*>.c
<*channel number*>: 0 or 1

RPDL function R_IIC_Destroy

Details
- Stops an I2C bus interface channel and places it in the module-stop state.

Example A case where the setting is made as follows.
- The function of master is selected for a RIIC0
- "Wait at the reception function until all data has been transmitted" is selected as the master reception method

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

4.21 Serial Peripheral Interface (RSPI)

4.21.1 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 the direction (input or output) and input buffer for the pin to be used is set.
- Function R_PG_Clock_Set must be called before calling this function.
- The commands are not set in this function. To set the commands, call R_PG_RSPI_SetCommand_C<*channel number*>.

Example: RSPI0 has been set in GUI.

```
//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
}
```

4.21.2 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

```
//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
}
```

4.21.3 R_PG_RSPI_StartTransfer_C<*channel number*>

Definition

• Transmit and receive

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

• Only transmit

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.

- RSPI0 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();
}
```

4.21.4 R_PG_RSPI_TransferAllData_C<*channel number*>

Definition

• Transmit and receive

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

• Only transmit

bool R_PG_RSPI_TransferAllData_C<*channel number*>
(uint32_t * tx_start, uint16_t sequence_loop_count)
<*channel number*>: 0, 1

• DTC/DMAC transfer

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.

- RSPI0 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_StartTransfer_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();
}
```

4.21.5 R_PG_RSPI_GetStatus_C<*channel number*>

Definition

bool R_PG_RSPI_GetStatus_C<*channel number*>
(bool * idle, bool * receive_full, bool * transmit_empty)
<*channel number*>: 0, 1

Description

Acquire the transfer status

Parameter

bool * idle	The address of storage area for the idle flag (0: Idle state 1: Transfer state)
bool * receive_full	The address of storage area for the receive buffer full flag (0: Empty 1: Full)
bool * transmit_empty	The address of storage area for the transmit buffer empty flag (0: Full 1: Empty)

Return value

true	Acquisition succeeded
false	Acquisition failed

File for output

R_PG_RSPI_C<*channel number*>.c
<*channel number*>: 0, 1

RPDL function

R_SPI_GetStatus

Details

- Acquires the transfer status.
- Specify the address of storage area for the items to be acquired. Specify 0 for an item that is not required.
- The error flags (the overrun error flag, the mode fault error flag, and the parity error flag) are cleared in this function. Call R_PG_RSPI_GetError_C<*channel number*> to acquire the error flags before calling this function if needed.

Example

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

bool idle;

void func(void)
{
      do{
            R_PG_RSPI_GetStatus_C0( &idle, 0, 0 );
      }while( idle );
}
```

4.21.6 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*>

4.21.7 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.

- RSPI0 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( &error_command );

        // Processing when an error is detected
    }
}
```

4.21.8 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*>.

4.21.9 R_PG_RSPI_LoopBack<*loopback mode*>_C<*channel number*>

Definition
bool R_PG_RSPI_LoopBack<*loopback mode*>_C<*channel number*> (void)
<*loopback mode*>: Direct, Reversed, Disable
<*channel number*>: 0, 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
}
```

4.22 12-Bit A/D Converter (S12AD)

4.22.1 R_PG_ADC_12_Set_S12AD0

Definition: bool R_PG_ADC_12_Set_S12AD0 (void)

Description: Set up the 12-Bit A/D Converter

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_ADC_12_S12AD0.c

RPDL function: R_ADC_12_Create

Details:

- Releases the 12-Bit A/D converter from the module-stop state, makes initial settings, and places it in the conversion-start trigger-input wait state. When the software trigger is selected to start conversion, conversion is started by calling R_PG_ADC_12_StartConversionSW_S12AD0.
- Function R_PG_Clock_Set must be called before calling this function.
- The input direction is set for pins used as analog inputs and the input buffers for the pins are disabled.
- The A/D-conversion end interrupt is set in this function. When the name of the interrupt notification function has been specified in the GUI, if an interrupt occurs in the CPU, the function having the specified name will be called. Create the interrupt notification function as follows:
  void <*name of the interrupt notification function*> (void)
  For the interrupt notification function, note the contents of this chapter end, Notes on Notification Functions.

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

- AN1 has been selected for the analog input chanel.
- S12ad0IntFunc has been specified as the A/D conversion end interrupt notification function name

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

uint16_t result[2];    // Storage for the result (AN0 to AN1 (AN0 is disable) )

void func(void)
{
      R_PG_Clock_Set();                 //Set up the clocks
      R_PG_ADC_12_Set_S12AD0();    //Set up the 12-Bit A/D converter (S12AD0)
}

//The A/D conversion end interrupt notification function
void S12ad0IntFunc(void)
{
      R_PG_ADC_12_GetResult_S12AD0(result);    //Acquire the result of A/D conversion
}
```

4.22.2 R_PG_ADC_12_StartConversionSW_S12AD0

Definition	bool R_PG_ADC_12_StartConversionSW_S12AD0(void)
Description	Start A/D conversion (Software trigger)
Conditions for output	Setting of the A/D converter and specification of the software trigger as the activation source
Parameter	None

Return value

true	Setting was made correctly
false	Setting failed

File for output	R_PG_ADC_12_S12AD0.c
RPDL function	R_ADC_12_Control

Details

- Starts A/D conversion by an A/D converter for which the software trigger is selected as the activation source.

Example

A case where the setting is made as follows.

- The software trigger is selected as the conversion start trigger.
- S12ad0IntFunc has been specified as the A/D conversion end 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_Clock_Set();                   //Set up the clocks
    R_PG_ADC_12_Set_S12AD0();       //Set up the 12-Bit A/D converter (S12AD0)

    // Start A/D conversion by the software trigger
    R_PG_ADC_12_StartConversionSW_S12AD0();
}
```

4.22.3 R_PG_ADC_12_StopConversion_S12AD0

Definition bool R_PG_ADC_12_StopConversion_S12AD0(void)

Description Stop A/D conversion

Parameter None

Return value

true	Stopping conversion succeeded.
false	Stopping conversion failed.

File for output R_PG_ADC_12_S12AD0.c

RPDL function R_ADC_12_Control

Details

- Stops A/D conversion in the continuous scan mode. In the one-cycle scan mode, this function need not be called after A/D conversion has ended.
  After this function has stopped A/D conversion, continuous scanning is resumed on input of the A/D-conversion start trigger. To end continuous scanning, stop the A/D conversion unit by calling R_PG_ADC_12_StopModule_S12AD0.

Example A case where the setting is made as follows.

- The continuous scan mode is selected as the operation mode.

```
//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_ADC_12_Set_S12AD0();           //Set up the 12-Bit A/D converter (S12AD0)
}

void func2(void)
{
    //Stop the continuous scan
    R_PG_ADC_12_StopConversion_S12AD0();
}
```

4.22.4 R_PG_ADC_12_GetResult_S12AD0

Definition: bool R_PG_ADC_12_GetResult_S12AD0(uint16_t * result)

Description: Acquire the result of A/D conversion

Parameter

uint16_t * result	The address of storage area for the result of A/D conversion

Return value

true	Acquisition of the result succeeded
false	Acquisition of the result failed

File for output: R_PG_ADC_12_S12AD0.c

RPDL function: R_ADC_12_Read

Details

- The amount of data to be acquired depends on the number of A/D-conversion channels that are in use. Reserve the area required for storing the result of A/D conversion for the given number of channels.
- When A/D conversion is in progress at the time of calling this function and a name for the interrupt notification function has not been specified through the GUI, the function waits until the end of A/D conversion before reading the result.

Example

A case where the setting is made as follows.

- AN0 to AN3 have been selected for the analog input chanels.
- S12ad0IntFunc has been specified as the A/D conversion end 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_Clock_Set();                   //Set up the clocks
    R_PG_ADC_12_Set_S12AD0();       //Set up the 12-Bit A/D converter (S12AD0)
}

//The A/D conversion end interrupt notification function
void S12ad0IntFunc(void)
{
    uint16_t result[4];    //Storage for the result of A/D conversion (AN0 to AN3)
    uint16_t result_an0;    //A/D Storage for the result of A/D conversion (AN0)
    uint16_t result_an1;    //A/D Storage for the result of A/D conversion (AN1)
    uint16_t result_an2;    //A/D Storage for the result of A/D conversion (AN2)
    uint16_t result_an3;    //A/D Storage for the result of A/D conversion (AN3)

    // Acquire the result of A/D conversion
    R_PG_ADC_12_GetResult_S12AD0( result );

    result_an0 = result[0];
    result_an1 = result[1];
    result_an2 = result[2];
    result_an3 = result[3];
}
```

4.22.5 R_PG_ADC_12_StopModule_S12AD0

Definition bool R_PG_ADC_12_StopModule_S12AD0(void)

Description Shut down the 12-Bit A/D converter

Parameter None

Return value

true	Shutting down succeeded
false	Shutting down failed

File for output R_PG_ADC_12_S12AD0.c

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[8];    //Storage for the result of A/D conversion (AN0 to AN7)

void func1(void)
{
      R_PG_Clock_Set();                 //Set up the clocks
      R_PG_ADC_12_Set_S12AD0();         //Set up the 12-Bit A/D converter (S12AD0)
}

void func2(void)
{
      //Stop the continuous scan
      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();
}
```

4.23 10-Bit A/D Converter (ADa)

4.23.1 R_PG_ADC_10_Set_AD<*unit number*>

Definition: bool R_PG_ADC_10_Set_AD<*unit number*> (void) <*unit number*>: 0 to 1

Description: Set up the 10-Bit A/D Converter (ADa)

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_ADC_10_AD<*unit number*>.c <*unit number*>: 0 to 1

RPDL function: R_ADC_10_Create

Details:

- Releases an A/D converter from the module-stop state, makes initial settings, and places it in the conversion-start trigger-input wait state. When the software trigger is selected to start conversion, conversion is started by calling R_PG_ADC_10_StartConversionSW_AD<*channel number*>.
- Function R_PG_Clock_Set must be called before calling this function.
- The input direction is set for pins used as analog inputs and the input buffers for the pins are disabled.
- The A/D-conversion end interrupt is set in this function. When the name of the interrupt notification function has been specified in the GUI, if an interrupt request is conveyed to the CPU, the function having the specified name will be called. Create the interrupt notification function as follows:

  void <name of the interrupt notification function> (void)

  For the interrupt notification function, note the contents of this chapter end, Notes on Notification Functions.

Example: AD2 has been set in the GUI.

Ad2IntFunc has been specified as the name of the A/D-conversion end interrupt notification function in the GUI.

```
#include "R_PG_default.h" //Include "R_PG_<PDG project name>.h" to use this function.
uint16_t data;    //Destination for storage of the result of A/D conversion

void func(void)
{
      R_PG_Clock_Set();                //The clock-generation circuit has to be set first.
      R_PG_ADC_10_Set_AD2();       //Set up AD2.
}

//AD-conversion end interrupt notification function
void Ad2IntFunc(void)
{
      R_PG_ADC_10_GetResult_AD2(&data)       //Acquire the result of A/D conversion.
}
```

4.23.2 R_PG_ADC_10_StartConversionSW_AD<*unit number*>

Definition: bool R_PG_ADC_10_StartConversionSW_AD<*unit number*> (void)
<*unit number*>: 0 to 1

Description: Start the A/D conversion (Software trigger)

Conditions for output: Setting of the A/D converter and specification of the software trigger as the activation source

Parameter: None

Return value:

true	Triggering the conversion succeeded.
false	Triggering the conversion failed.

File for output: R_PG_ADC_10_AD<*unit number*>.c
<*unit number*>: 0 to 1

RPDL function: R_ADC_10_Control

Details:

- Starts A/D conversion by an A/D converter for which the software trigger is selected as the activation source.

Example: The continuous scan mode has been specified as the AD2 mode 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_AD2();         //Set up AD2.

    //Start A/D conversion by the software trigger
    R_PG_ADC_10_StartConversionSW_AD2();
}
```

4.23.3 R_PG_ADC_10_StopConversion_AD<*unit number*>

Definition bool R_PG_ADC_10_StopConversion_AD<*unit number*> (void)
<*unit number*>: 0 to 1

Description Stop the A/D conversion

Parameter None

Return value

true	Stopping the conversion succeeded.
false	Stopping the conversion failed.

File for output R_PG_ADC_10_AD<*unit number*>.c
<*unit number*>: 0 to 1

RPDL function R_ADC_10_Control

Details

- Stops A/D conversion in the continuous scan mode. In the single mode and single-cycle scan mode, this function need not be called after A/D conversion has ended.
  After this function has stopped A/D conversion, continuous scanning is resumed on input of the A/D-conversion start trigger. To end continuous scanning, stop the A/D conversion unit by calling R_PG_ADC_10_StopModule_AD<*unit number*>.

Example The software trigger has been specified as the activation source for AD2 in the GUI.

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

uint16_t data;    //Destination for storage of the result of A/D conversion

void func1(void)
{
      R_PG_Clock_Set();                //The clock-generation circuit has to be set first.
      R_PG_ADC_10_Set_AD2();       //Set up AD2.
}

void func2(void)
{
      //Stop continuous scanning.
      R_PG_ADC_10_StopConversion_AD2();

      //Acquire the result of A/D conversion.
      R_PG_ADC_10_GetResult_AD2(&data)
}
```

4.23.4 R_PG_ADC_10_GetResult_AD<*unit number*>

Definition: bool R_PG_ADC_10_GetResult_AD<*unit number*> (uint16_t * result)

<*unit number*>: 0 to 1

Description: Get the result of A/D conversion

Parameter

uint16_t * result	Destination for storage of the result of A/D conversion

Return value

true	Acquisition of the result succeeded.
false	Acquisition of the result failed.

File for output: R_PG_ADC_10_AD<*unit number*>.c <*unit number*>: 0 to 1

RPDL function: R_ADC_10_Read

Details

- The amount of data to be acquired depends on the number of A/D-conversion channels that are in use. Reserve the area required for storing the result of A/D conversion for the given number of channels.
- When A/D conversion is in progress at the time of calling this function and a name for the interrupt notification function has not been specified through the GUI, the function waits until the end of A/D conversion before reading the result.

Example

The single-cycle scan mode has been specified for AD0 in in the GUI.
Four channels (AN0 to AN3) are in use.
Ad0IntFunc has been specified as the name of the A/D-conversion end interrupt notification function in the GUI.

```
//Include "R_PG_<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 AD0.
}
//AD-conversion end interrupt notification function
void Ad0IntFunc(void)
{
    uint16_t data[4];    //Result of A/D conversion on all channels
    uint16_t data_an0;   //Result of A/D conversion on AN0
    uint16_t data_an1;   //Result of A/D conversion on AN1
    uint16_t data_an2;   //Result of A/D conversion on AN2
    uint16_t data_an3;   //Result of A/D conversion on AN3

    R_PG_ADC_10_GetResult_AD0(data)     //Acquire the results of A/D conversion.

    data_an0 = data[0];
    data_an1 = data[1];
    data_an2 = data[2];
    data_an3 = data[3];
}
```

4.23.5 R_PG_ADC_10_SetSelfDiag_VREF_<*voltage*>_AD<*unit number*>

Definition: bool R_PG_ADC_10_SetSelfDiag_VREF_<*voltage*>_AD<*unit number*> (void)
<*voltage*>: 0, 0_5, 1 (0:Vref*0, 0_5:Vref/2, 1:Vref) <*unit number*>: 0 to 1

Description: Set up the A/D self-diagnostic function

Conditions for output: The self-diagnostic function is enabled

Parameter: None

Return value:

true	Setting was made correctly
false	Setting failed

File for output: R_PG_ADC_10_AD<*unit number*>.c <*unit number*>: 0 to 1

RPDL function: R_ADC_10_Create

Details:

- Setsup the A/D self-diagnostic function.
- In this function, the A/D conversion mode is set to the single mode and the conversion start trigger is set to the software trigger. To re-set the A/D converter, call R_PG_ADC_10_Set_AD<*unit number*>.
- To start the self-diagnostic, call R_PG_ADC_10_StartSelfDiag_AD<*unit number*> and to get the result of self-diagnostic, call R_PG_ADC_10_GetResult_AD<*unit number*>.

Example:

```
//Include "R_PG_<PDG project name>.h" to use this function.
#include "R_PG_default.h"
uint16_t SelfDiagnostic_0(void)
{
    uint16_t result;
    R_PG_ADC_10_SetSelfDiag_VREF_0_AD0();
    R_PG_ADC_10_StartSelfDiag_AD0();
    R_PG_ADC_10_GetResult_AD0 (&result);
    return result;
}
uint16_t SelfDiagnostic_0_5(void)
{
    uint16_t result;
    R_PG_ADC_10_SetSelfDiag_VREF_0_5_AD0();
    R_PG_ADC_10_StartSelfDiag_AD0();
    R_PG_ADC_10_GetResult_AD0 (&result);
    return result;
}
uint16_t SelfDiagnostic_1(void)
{
    uint16_t result;
    R_PG_ADC_10_SetSelfDiag_VREF_1_AD0();
    R_PG_ADC_10_StartSelfDiag_AD0();
    R_PG_ADC_10_GetResult_AD0 (&result);
    return result;
}
```

4.23.6 R_PG_ADC_10_StartSelfDiag_AD<*unit number*>

Definition	bool R_PG_ADC_10_StartSelfDiag_AD<*unit number*> (void) <*unit number*>: 0 to 1
Description	Start the A/D conversion (Self-diagnostic function)
Conditions for output	The self-diagnostic function is enabled
Parameter	None

Return value

true	Triggering the conversion succeeded
false	Triggering the conversion failed

File for output	R_PG_ADC_10_AD<*unit number*>.c <*unit number*>: 0 to 1
RPDL function	R_ADC_10_Control

Details

- Starts the A/D conversion of the self-diagnostic function.
- Call R_PG_ADC_10_SetSelfDiag_VREF_<*voltage*>_AD<*unit number*> to set up self-diagnostic function before calling this function.
- To get the result of self-diagnostic, call R_PG_ADC_10_GetResult_AD<*unit number*>.

Example Refer to the example of R_PG_ADC_10_SetSelfDiag_VREF_<*voltage*>_AD<*unit number*>

4.23.7 R_PG_ADC_10_StopModule_AD<*unit number*>

Definition bool R_PG_ADC_10_StopModule_AD<*unit number*> (void)
<*unit number*>: 0 to 1

Description Shut down the 10-Bit A/D Converter (ADa)

Parameter None

Return value

true	Shutting down succeeded
false	Shutting down failed

File for output R_PG_ADC_10_AD<*unit number*>.c
<*unit number*>: 0 to 1

RPDL function R_ADC_10_Destroy

Details
- Stops an 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 data;    //Destination for storage of the result of A/D conversion

void func1(void)
{
      R_PG_Clock_Set();                 //The clock-generation circuit has to be set first.
      R_PG_ADC_10_Set_AD2();        //Set up AD2.
}

void func2(void)
{
      //Stop continuous scanning.
      R_PG_ADC_10_StopConversion_AD2();

      //Acquire the result of A/D conversion.
      R_PG_ADC_10_GetResult_AD2(&data)

      //Stop the A/D converter.
      R_PG_ADC_10_StopModule_AD2();
}
```

4.24 D/A Converter

4.24.1 R_PG_DAC_Set_C<*channel number*>

Definition: bool R_PG_DAC_Set_C<*channel number*> (void) <*channel number*>: 0 or 1

Description: Set up a 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 or 1

RPDL function: R_DAC_10_Create

Details:
- Sets up a D/A converter channel and enables the analog output.
- D/A converter shall be released from the module-stop state.

Limitations in evaluation version:
- Both DA0 and DA1 cannot be used in the same time.
- GUI and function specification may be changed in the MP version.

Example:

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

void func1(void)
{
      //Set up DA0
      R_PG_DAC_Set_C0();
}

void func2( uint16_t output_val )
{
      //Input data
      R_PG_DAC_ControlOutput_C0( output_val );
}
```

4.24.2 R_PG_DAC_SetWithInitialValue_C<*channel number*>

Definition: bool R_PG_DAC_SetWithInitialValue_C<*channel number*> (uint16_t data)

<*channel number*>: 0 or 1

Description: Set up a D/A converter channel and input the data

Parameter

uint16_t data	The value to be written to the output register

Return value

true	Setting was made correctly
false	Setting failed

File for output: R_PG_DAC_C<*channel number*>.c <*channel number*>: 0 or 1

RPDL function: R_DAC_10_Create

Details
- Sets up a D/A converter channel and enables the analog output.
- If another channel has not been used, D/A converter shall be released from the module-stop state.

Limitations in evaluation version
- Both DA0 and DA1 cannot be used in the same time.
- GUI and function specification may be changed in the MP version.

Example

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

void func1(uint16_t initial_val)
{
      //Set up DA0
      R_PG_DAC_SetWithInitialValue_C0( initial_val );
}

void func2( uint16_t output_val )
{
      //Write data to the
      R_PG_DAC_ControlOutput_C0( output_val );
}
```

4.24.3 R_PG_DAC_ControlOutput_C<*channel number*>

Definition: bool R_PG_DAC_ControlOutput_C<*channel number*> (uint16_t data)

<*channel number*>: 0 or 1

Description: Input the data

Parameter

uint16_t data	The value to be written to the D/A 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 or 1

RPDL function: R_DAC_10_Write

Details

- Sets the data to the D/A data register

Limitations in evaluation version

- GUI and function specification may be changed in the MP version.

Example

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

void func1(uint16_t initial_val)
{
      //Set up DA0
      R_PG_DAC_SetWithInitialValue_C0( initial_val );
}

void func2( uint16_t output_val )
{
      //Write data to the
      R_PG_DAC_ControlOutput_C0( output_val );
}
```

4.24.4 R_PG_DAC_StopOutput_C<*channel number*>

Definition: bool R_PG_DAC_StopOutput_C<*channel number*> (void)

<*channel number*>: 0 or 1

Description: Stop the analog output.

Parameter: None

Return value:

true	Stopping succeeded
false	Stopping failed

File for output: R_PG_DAC_C<*channel number*>.c <*channel number*>: 0 or 1

RPDL function: R_DAC_10_Destroy

Details:

- Disables D/A converter channel.

  Once both channels are disabled, the module is put into the power-down state.

Limitations in evaluation version:

- GUI and function specification may be changed in the MP version.

Example:

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

void func1(uint16_t initial_val)
{
      //Set up DA0
      R_PG_DAC_SetWithInitialValue_C0( initial_val );
}

void func2(void)
{
      //Stop the analog output
      R_PG_DAC_StopOutput_C0();
}
```

4.24.5 R_PG_DAC_Set_C0_C1

Definition	bool R_PG_DAC_Set_C0_C1 (void)
Description	Set up the D/A converter channel (DA0 and DA1)
Conditions for output	Both DA0 and DA1 are used.
Parameter	None

Return value

true	Setting was made correctly.
false	Setting failed.

File for output	R_PG_DAC.c
RPDL function	R_DAC_10_Create

Details

- Sets up a D/A converter channel (DA0 and DA1).
- Releases 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*>.

Example

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

void func1(void)
{
      //Setting up DA0 pin and DA1 pin.
      R_PG_DAC_Set_C0_C1();
}

void func2( uint16_t output_val_c0, uint16_t output_val_c1 )
{
      //Change D/A conversion value
      R_PG_DAC_ControlOutput_C0( output_val_c0 );
      R_PG_DAC_ControlOutput_C1( output_val_c1 );
}
```

4.25 Notes on Notification Functions

4.25.1 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.

4.25.2 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.

5. 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.

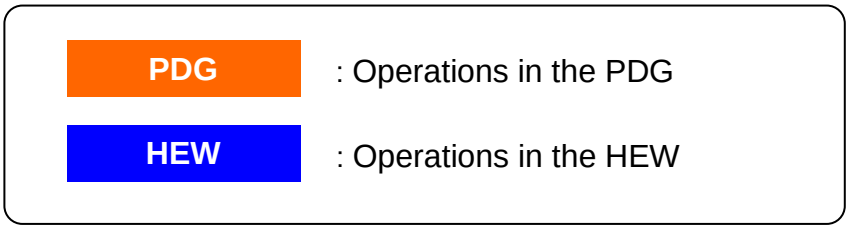
1. Select the project on Project Explorer.
2. Select [File]->[Properties] from main menu to open the [Properties] window.
3. Select [C/C++ build] ->[Settings]
4. Select [All configurations] for [Configuration]
5. Select [Linker] -> [Section] to show [Section viewer]
6. Set the address of the FIXEDVECT section as 0xFFFFFFD0.

6. Tutorial

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 RX62N.

- An LED blinking on a 8-bit timer (TMR) interrupt
- An LED blinking on the PWM output of the multi-function timer pulse unit 2 (MTU2)
- Continuously scanning on10-Bit A/D converter (ADa)
- Triggering DTC by IRQ
- Data transfer between SCIa channels 2 and 5

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

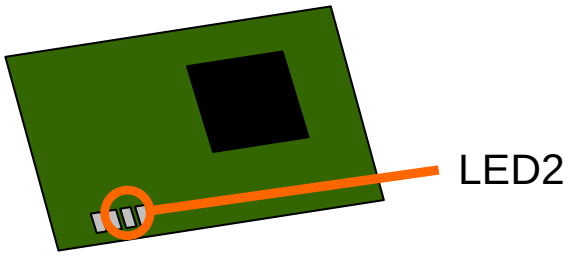


6.1 An LED blinking on a 8-bit timer (TMR) interrupt

The LED2 on RSK+ board is connected to P05. In this tutorial, 8-bit Timer and I/O port will be set up to blink this LED as follows.

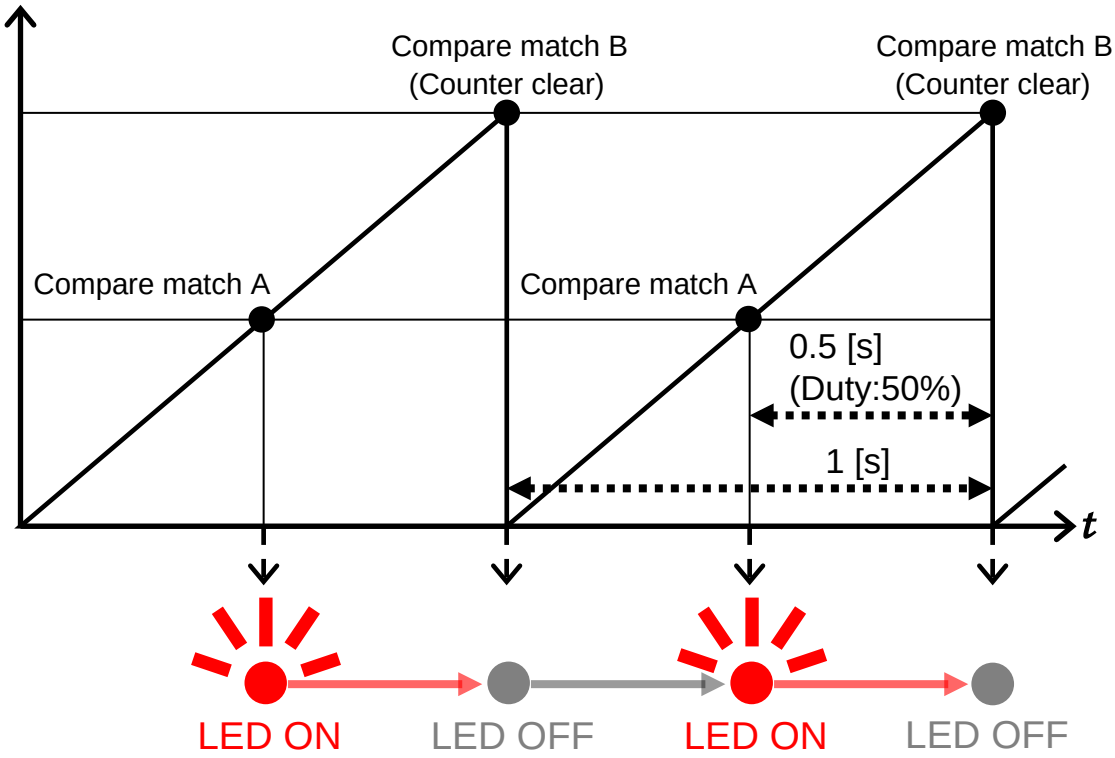
Note : If there is a switch that enables/disables P05 on the RSK+ board, enable it.

The LED2 turns on when the output from P05 is 0, and turns off when the output is 1.



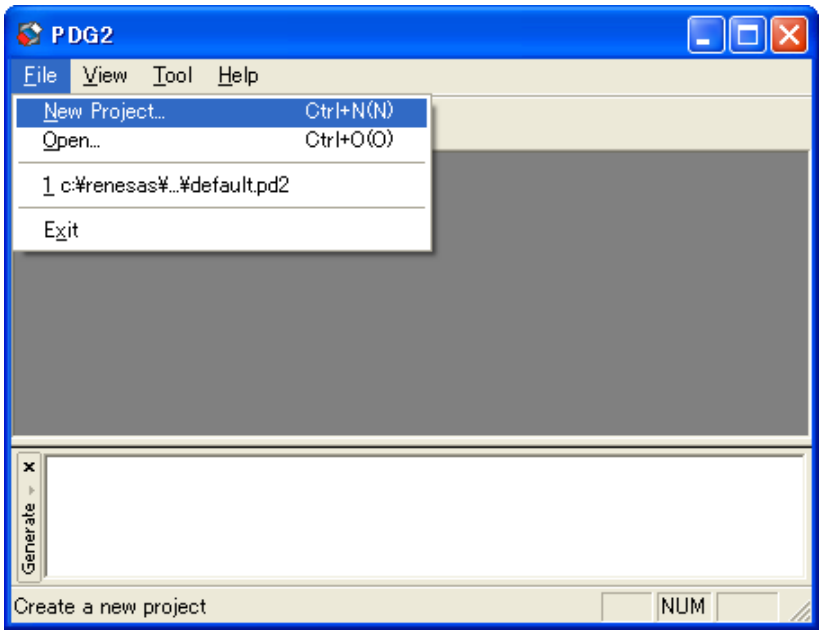
- Turn on the LED
  at compare match A
- Turn off the LED
  at compare match B
- Clear the counter
  at compare match B

TMR counter value



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

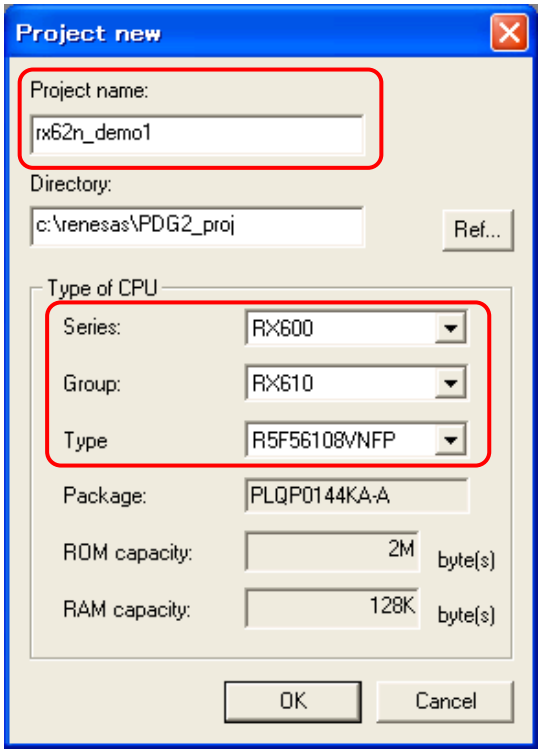
1. Start the PDG.
2. Select [File]->[New Project] menu.



3. Specify "rx62n_demo1" as the project name.
   Set the CPU type as follows.

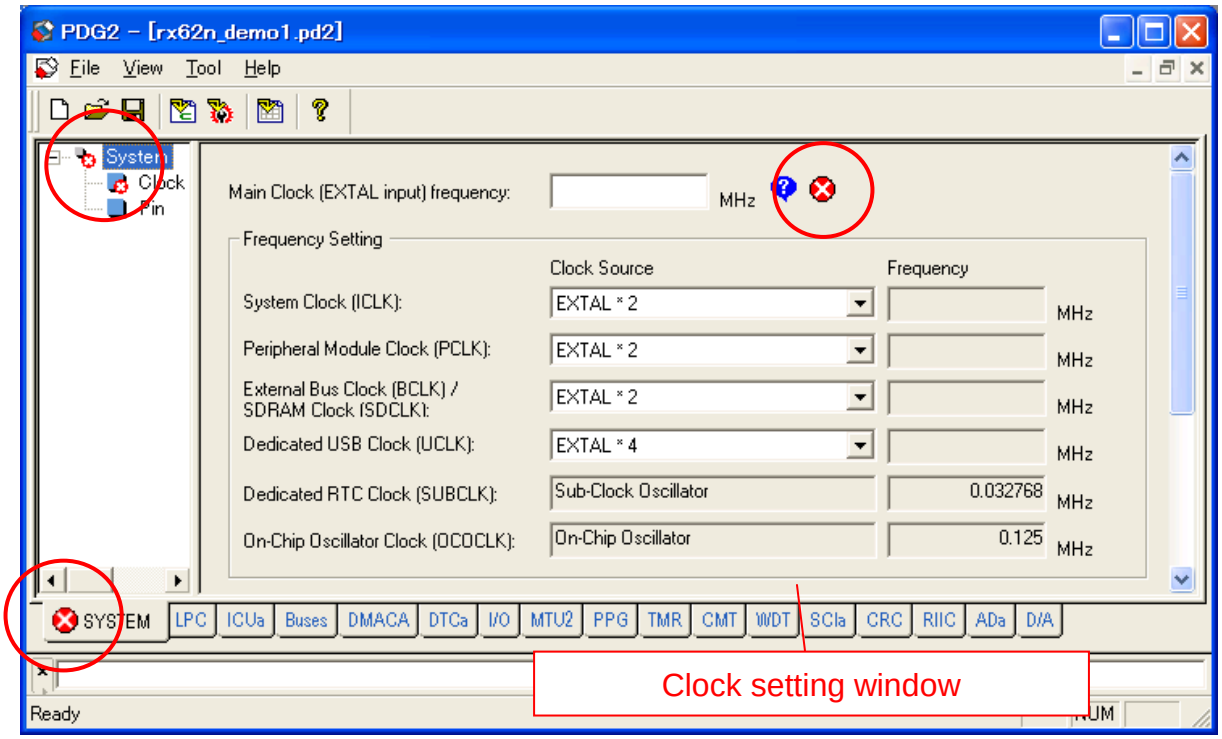
   Series : RX600
   Group : RX62N
   Type : R5F562N8BDBG

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

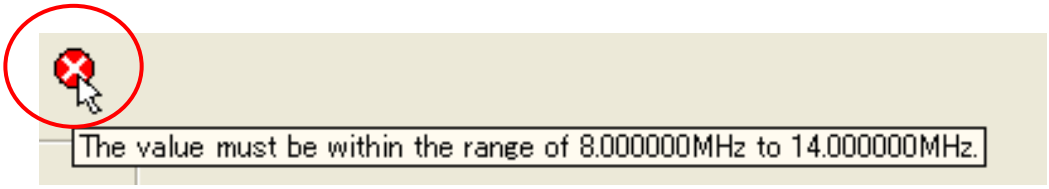


(2) Initial state **PDG**

-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.

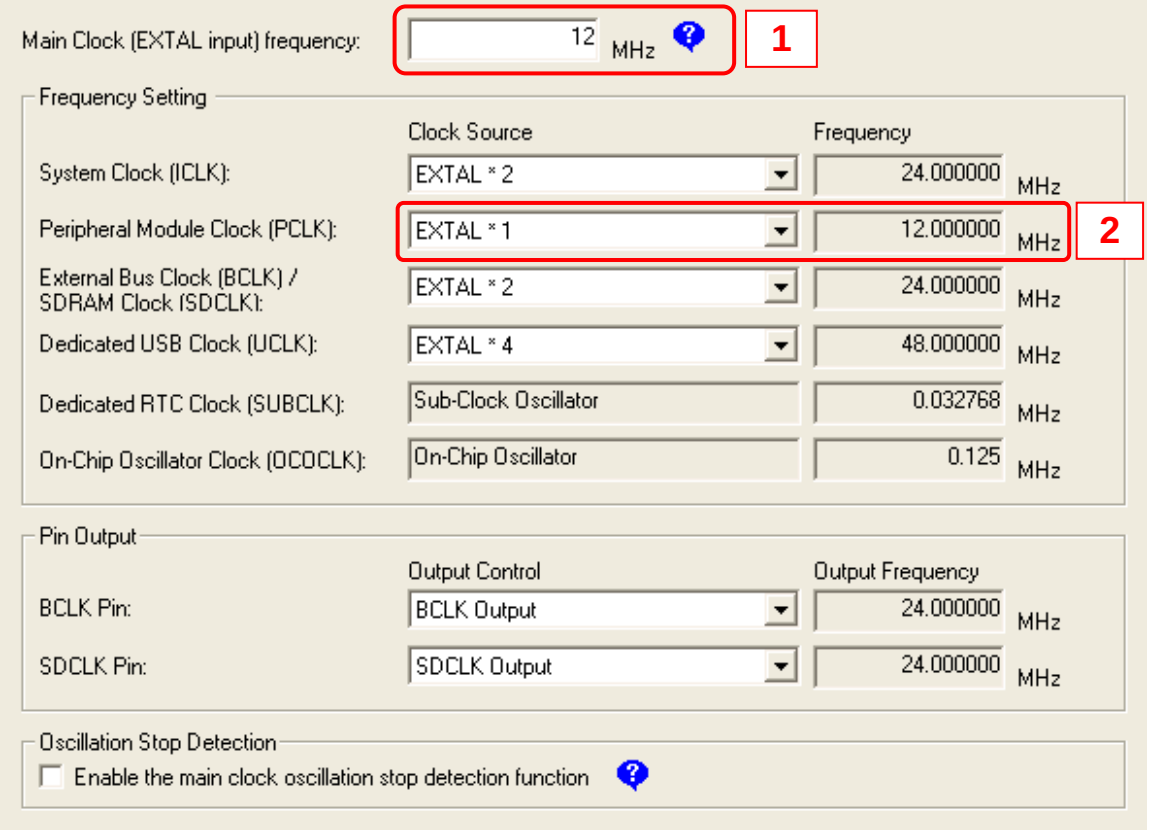
Warning
- The setting is possible but may be wrong.
- Source files can be generated.

Information
- Additional information for the complex setting.

Only icons on the setting window can display the tooltip.

(3) Clock setting **PDG**

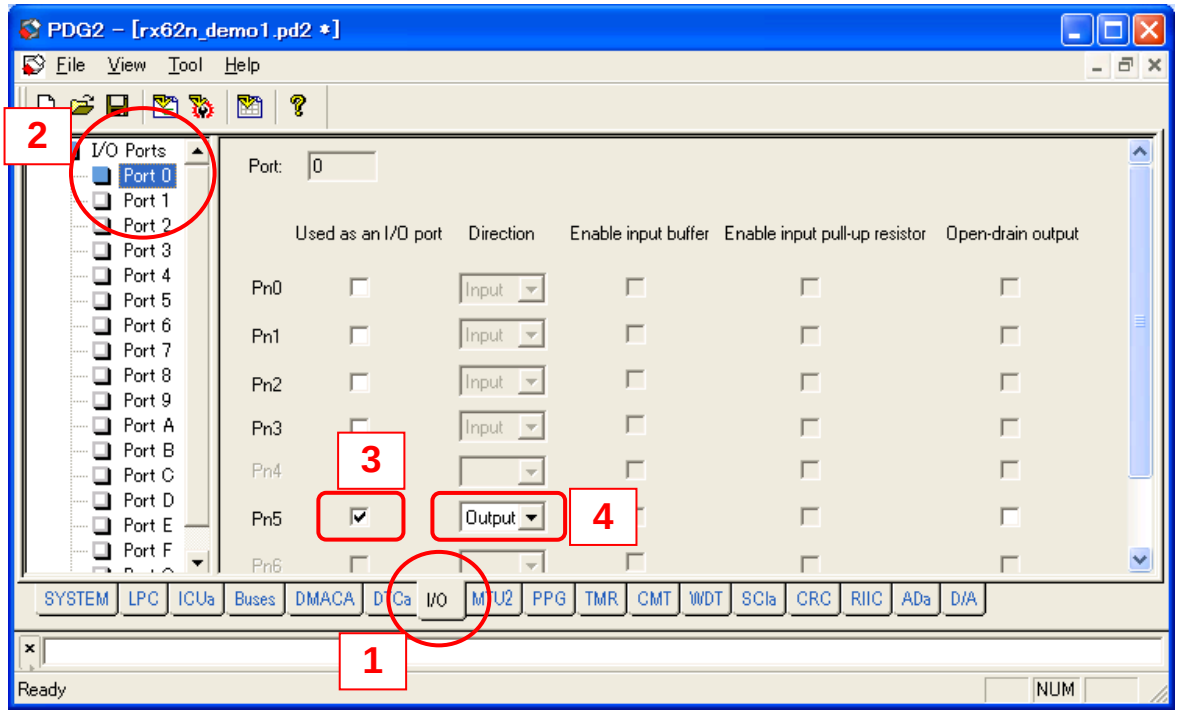
1. It is necessary to set the main (EXTAL) clock frequency first.
   External clock frequency of the RSK+ board is 12 MHz. Input “12” into the edit box.
2. PCLK is used in 12MHz.
   Select the multiplication "EXTAL x 1" to set the PCLK to 12MHz.



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

The LED1 on RSK+ is connected to P05 so set P05 to output port.

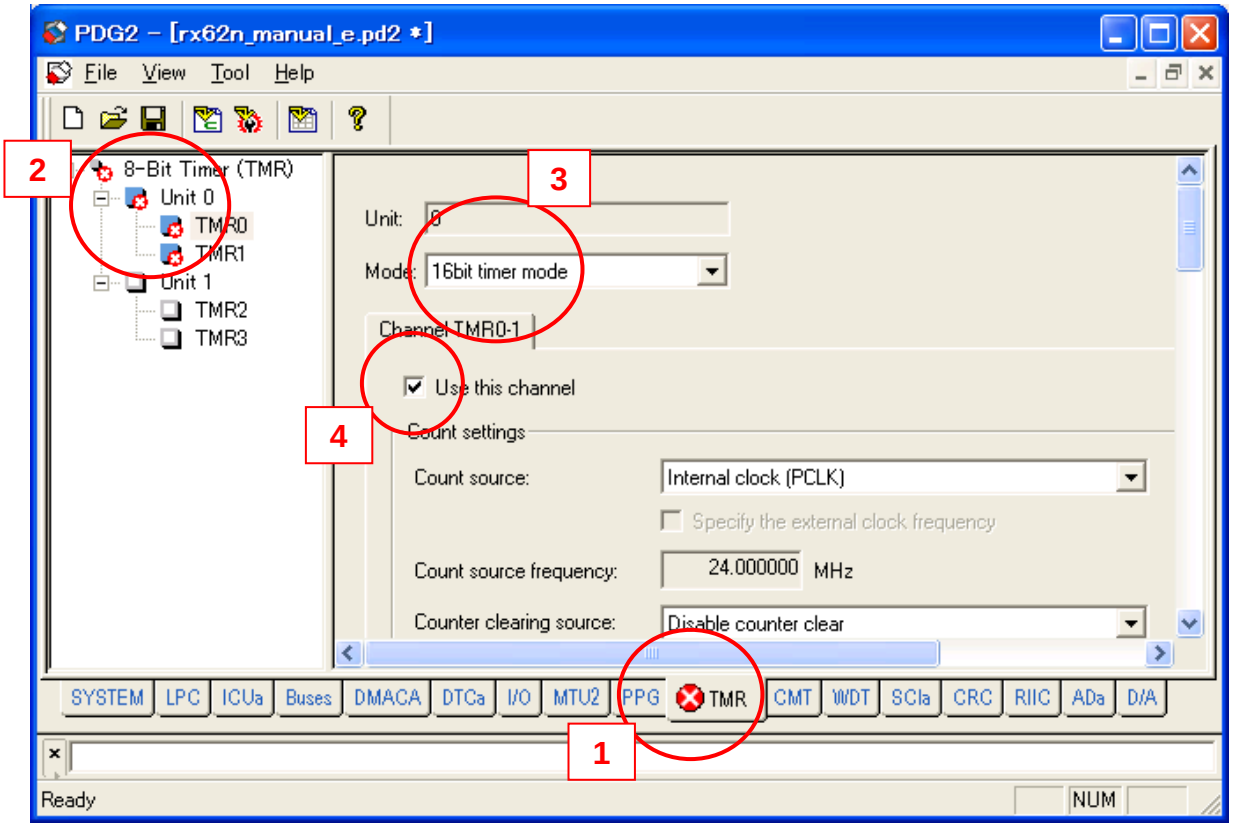
1. Select “I/O” tab
2. Select “Port 0”
3. Check “Pn5”
4. Select “Output”



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

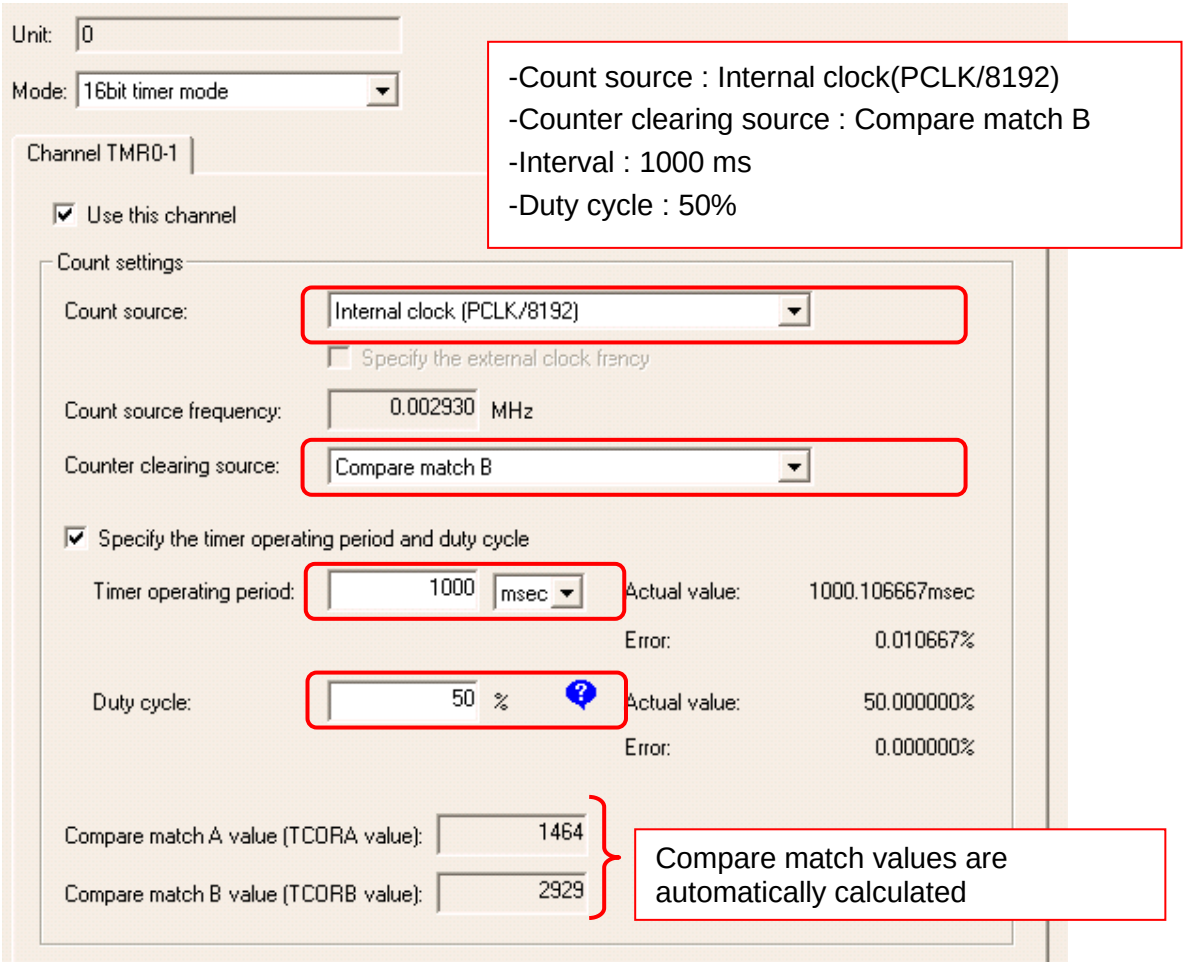
In this tutorial, TMR (8-bit timer) Unit0 is used in 16 bit mode (two 8-bit timers cascade connection)

1. Select "TMR" tab
2. Select "Unit0"
3. Select "16 bit timer mode"
4. Check "Use this channel"



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

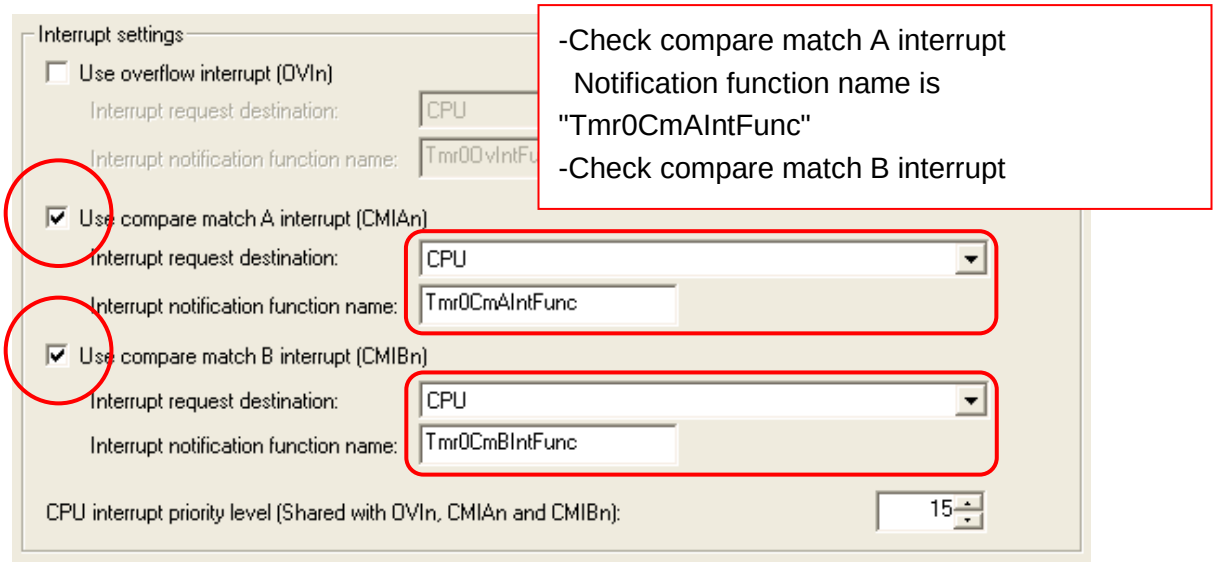
Set the other items as follows.



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

Set the interrupt notification functions.

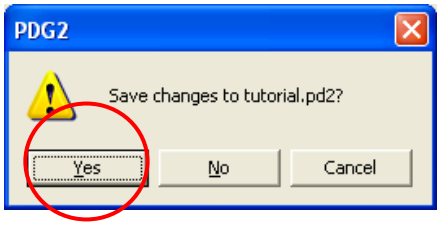
These functions are called when the interrupt occurs.



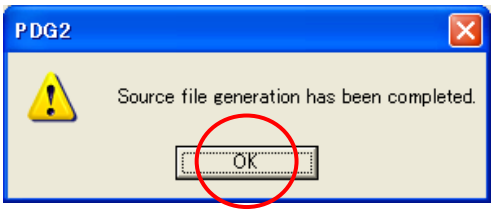
(8) Generating source files PDG

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

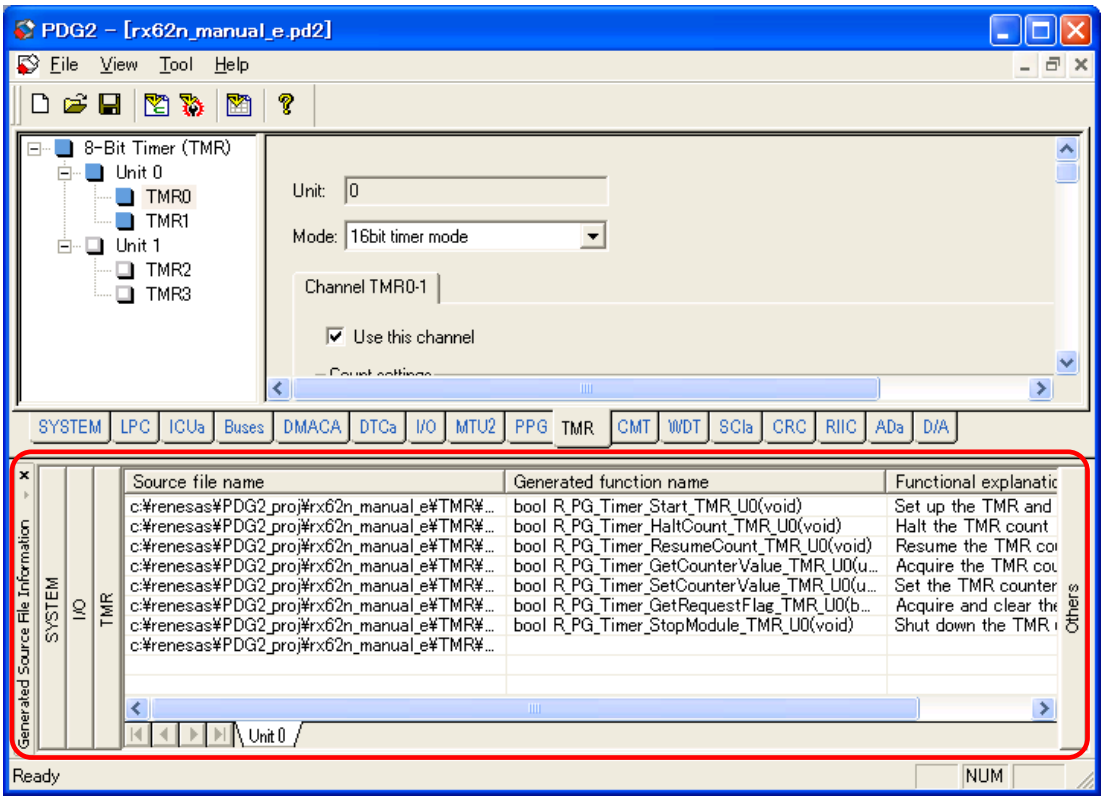
2. Save confirmation dialog box is displayed. Click [OK].



3. Click [OK] on the message box.

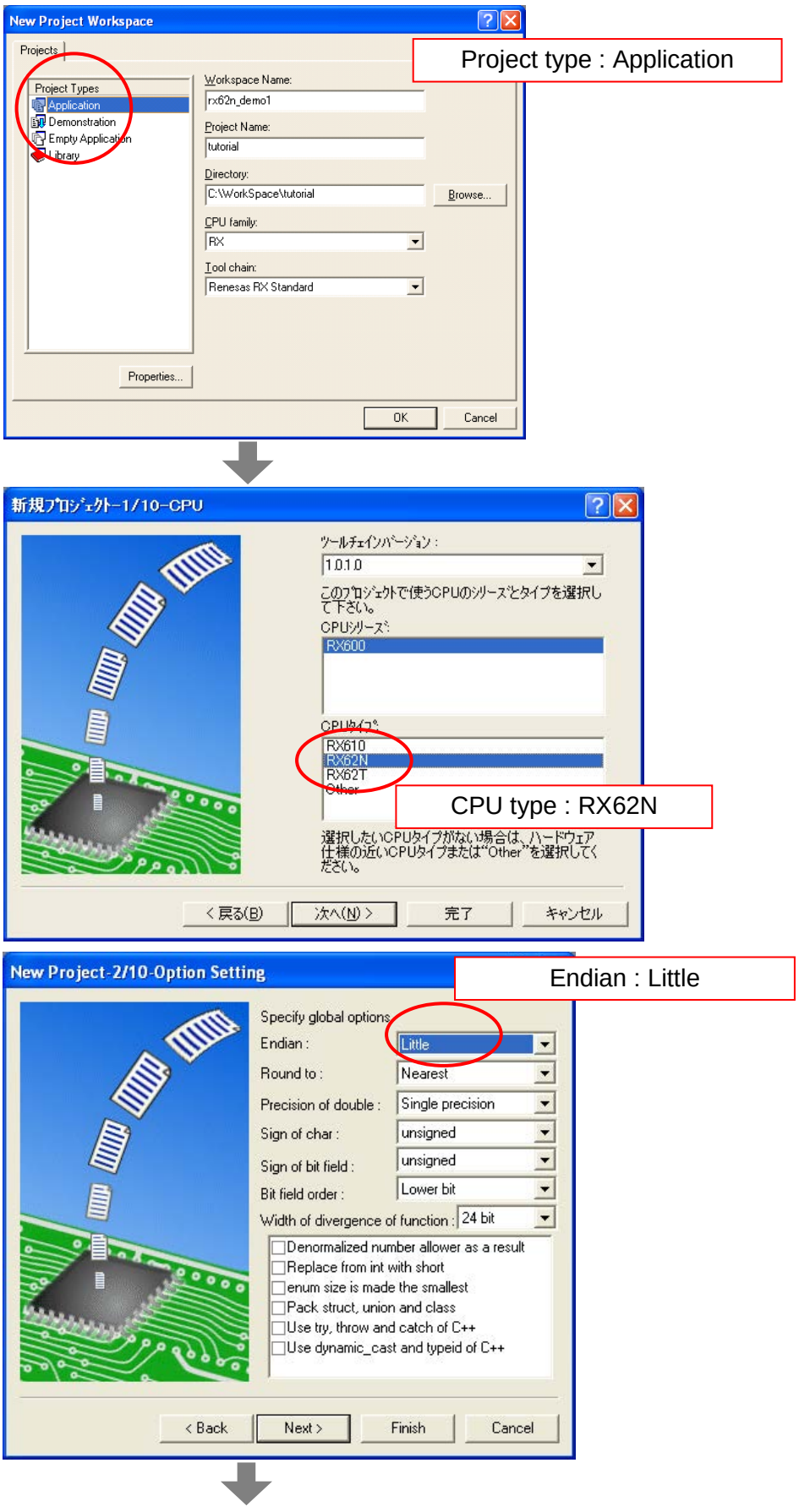


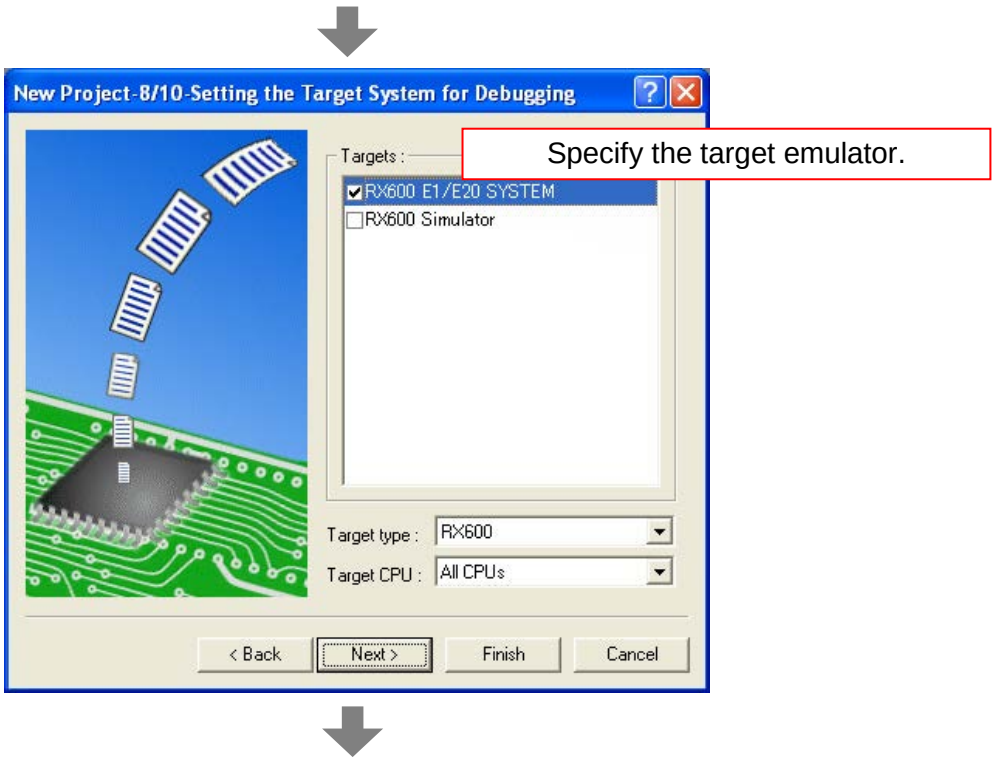
4. Generated functions are listed in lower pane.
   By double clicking the line of function, source file can be opened.

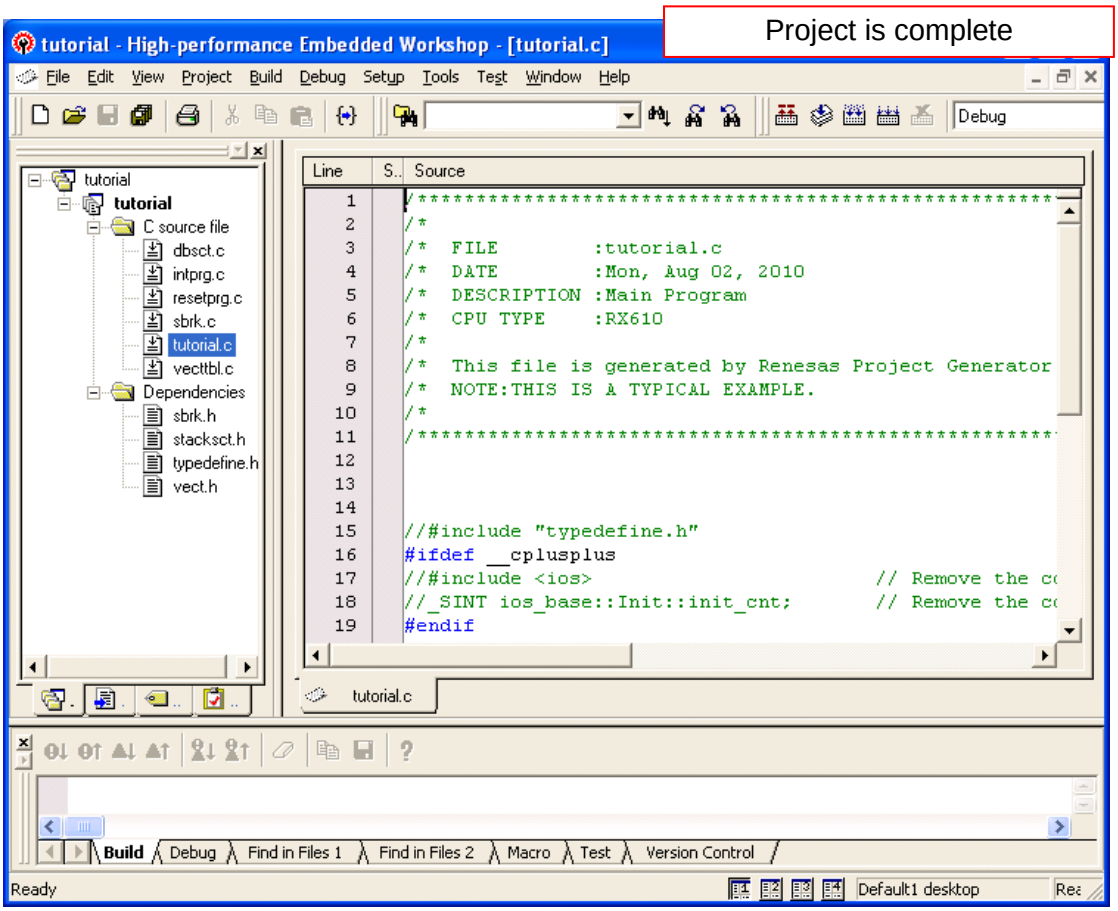


(9) Preparing the HEW project **HEW**

Start the HEW and make RX62N workspace.




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

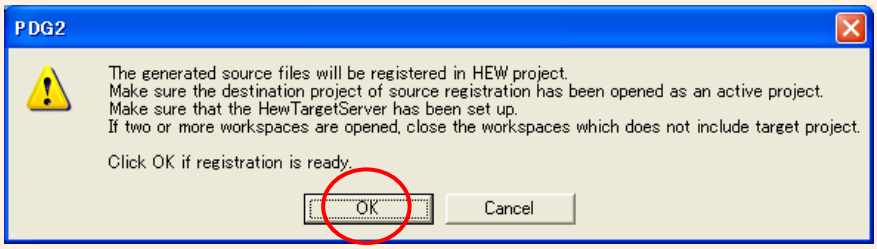

Project is complete
tutorial - High-performance Embedded Workshop - [tutorial.c]
File Edit View Project Build Debug Setup Tools Test Window Help
Debug
tutorial
C source file
dbsct.c
intprg.c
resetprg.c
sbrk.c
tutorial.c
vecttbl.c
Dependencies
sbrk.h
stacksct.h
typedefine.h
vect.h
Line S.. Source
1 /*
2 /* FILE :tutorial.c
3 /* DATE :Mon, Aug 02, 2010
4 /* DESCRIPTION :Main Program
5 /* CPU TYPE :RX610
6 /*
7 /* This file is generated by Renesas Project Generator
8 /* NOTE:THIS IS A TYPICAL EXAMPLE.
9 /*
10
11
12
13 //#include "typedefine.h"
14 #ifdef __cplusplus
15 //#include <ios>
16 //_SINT ios_base::Init::init_cnt;
17 #endif
tutorial.c
Build Debug Find in Files 1 Find in Files 2 Macro Test Version Control
Ready
Default1 desktop

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

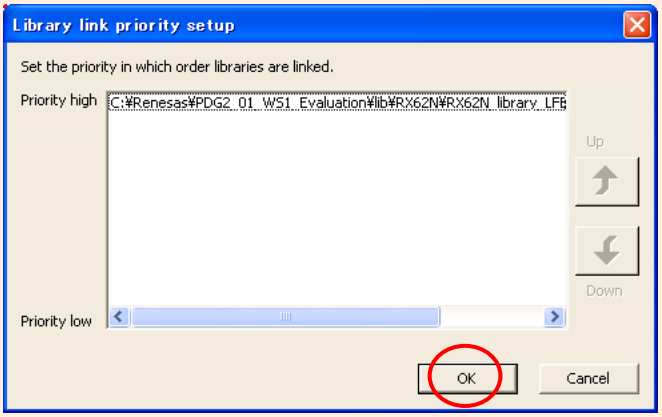
PDG

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

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

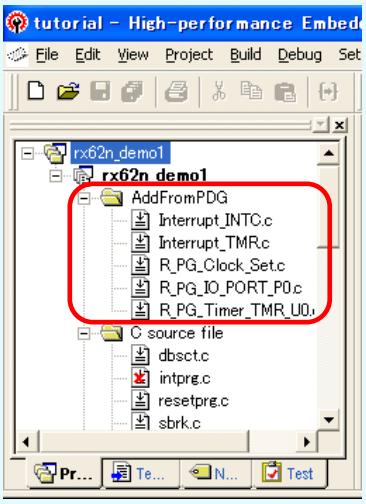


3. This is a linkage setting of RPDL library.
When using multiple lib files, linkage order can be set in this dialog box.



HEW

4. Source fiels 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.

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

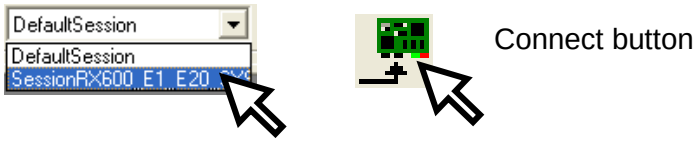
By changing the part of “main” function, make the following program on HEW.

```
//Include "R_PG_<PDG project name>.h"
#include "R_PG_rx62n_demo1.h"
void main(void)
{
	//Set up the clock
	R_PG_Clock_Set();
	//Set up port P05
	R_PG_IO_PORT_Set_P0();
	//Set up TMR Unit0 and start count
	R_PG_Timer_Start_TMR_U0();
	while(1);
}
// Compare match A interrupt notification function
void Tmr0CmAIntFunc(void)
{
	// Turn on the LED
	R_PG_IO_PORT_Write_P05(0);
}
// Compare match B interrupt notification function
void Tmr0CmBIntFunc(void)
{
	// Turn off the LED
	R_PG_IO_PORT_Write_P05(1);
}
```

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

1. Before connecting the emulator, make sure that SW4/Pin3 on RSK+ board is “ON” to set CPU to little endian.

2. Connect to the emulator

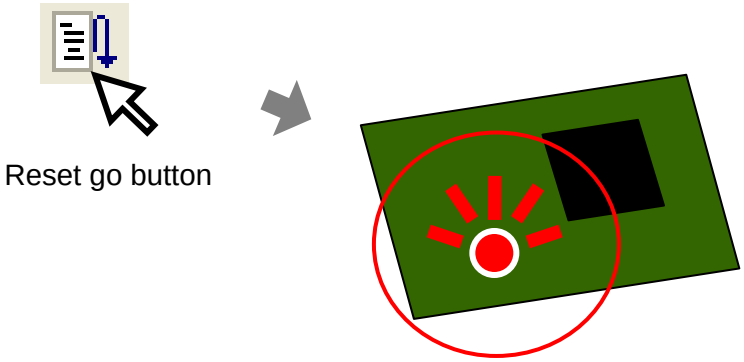


3. Just by clicking [Build] button, program can be built because RPDL library and include directory are automatically registered in build setting.

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

4. Download the program

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

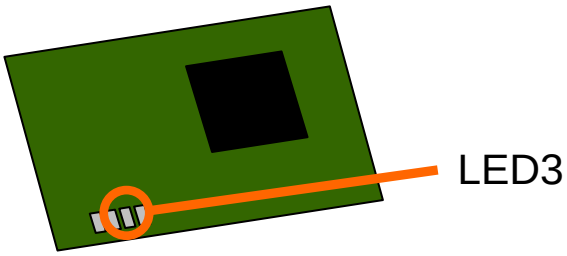


6.2 An LED blinking on the PWM output of the multi-function timer pulse unit 2 (MTU2)

The LED3 on RSK+ board is connected to P35. This port can alsp be used as PWM output pin (MTIOC0A) of the multi-function timer pulse unit 2. In this tutorial, the multi-function timer pulse unit 2 will be set up to operate in PWM mode 1 and the PWM output will blink the LED3 as follows.

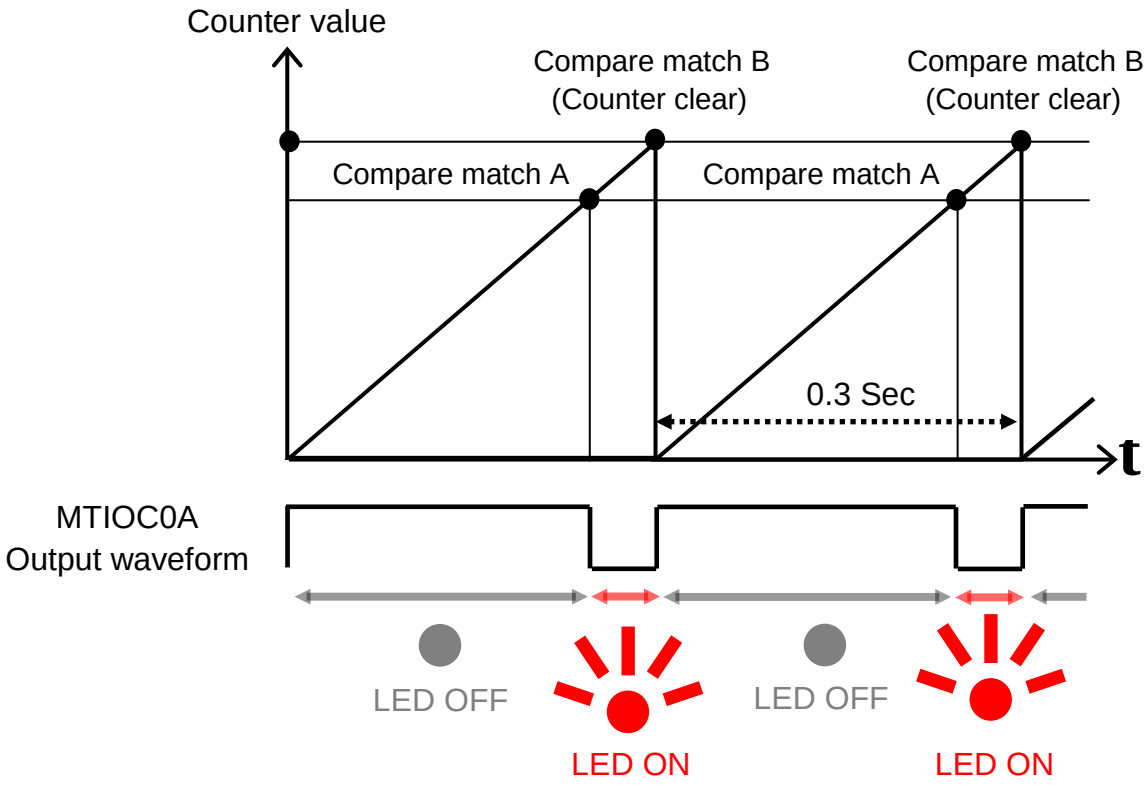
Note : If there is a switch that enables/disables P34(MTIOC0A) on the RSK+ board, enable it.

The LED2 turns on when the output from P34 is 0, and turns off when the output is 1.



The MTU2 channel 0 (MTU0) will be operated in PWM mode 1. In PWM mode 1, the output signal is controlled by compare match A and B.

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

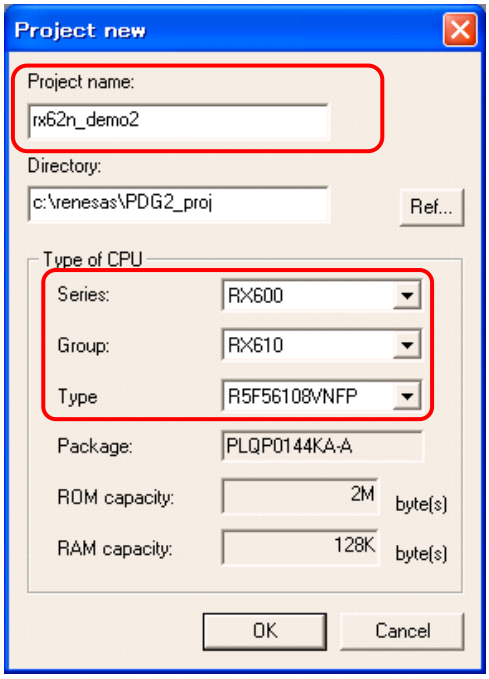


(1) Making the PDG project **PDG**

Make the new PDG project “rx62n_demo2”. For details on how to make the new PDG project, refer to section 6.1 (1), Making the PDG project.
Set the CPU type as follows.

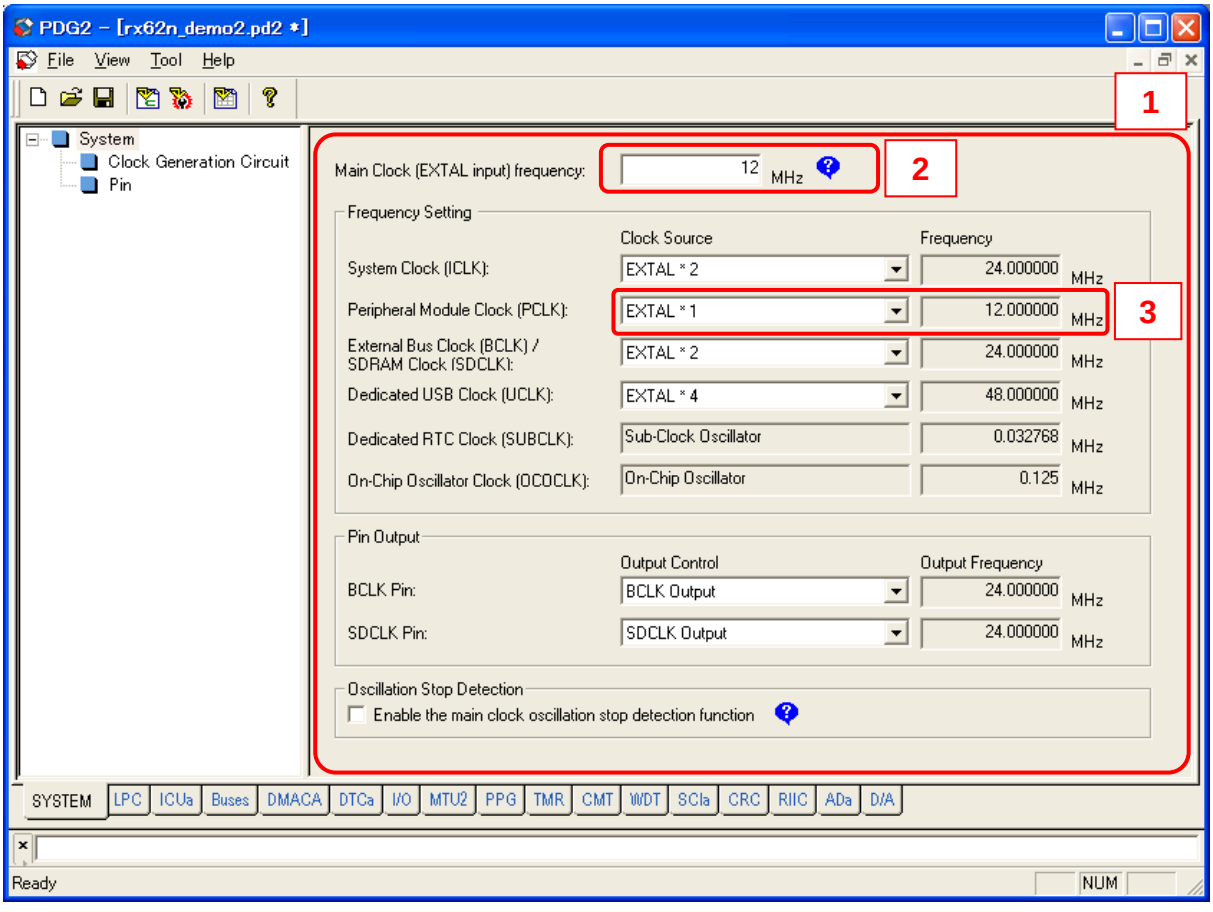
Series : RX600
Group : RX62N
Type : R5F562N8BDBG

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



(2) Clock setting **PDG**

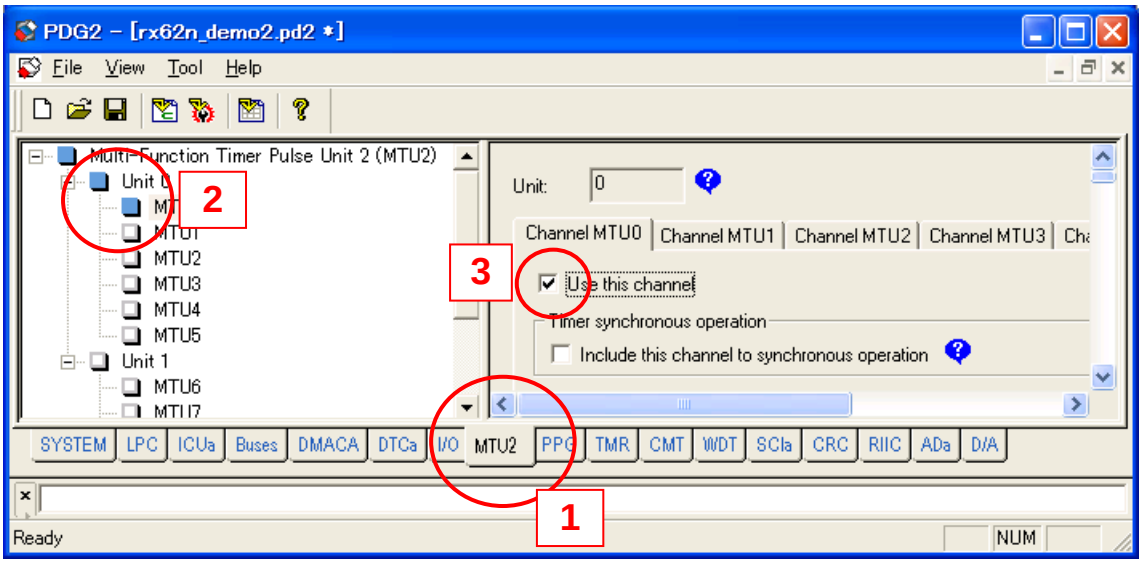
1. The clock setting window opens and the error icons are displayed in the initial state. For icons such as and displayed on window, refer to section 6.1 (2), Initial state.
2. External clock frequency of the RSK+ board is 12 MHz. Input “12” into the edit box.
3. PCLK is used in 12MHz. Select the multiplication "EXTAL x 1" to set the PCLK to 12MHz



(3) MTU2 setting-1 **PDG**

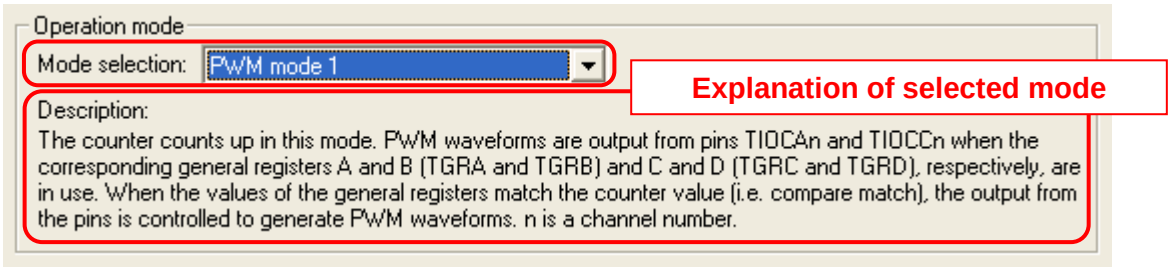
Opening MTU2 channel 0(MTU0) setting window

1. Select "MTU2" tab.
2. Select " MTU0" on tree view.
3. Check "Use this channel".



(4) MTU2 setting-2 **PDG**

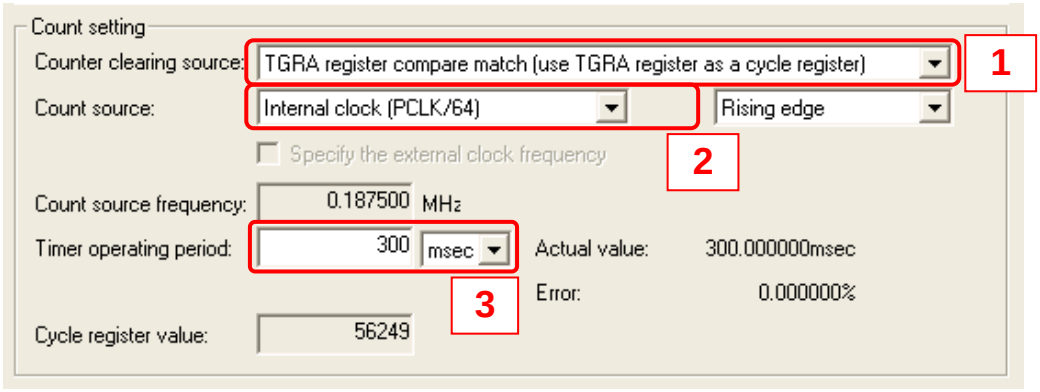
Select "PWM mode 1" for the operation mode.



(5) MTU2 setting-3 **PDG**

The counter setting is as follows.

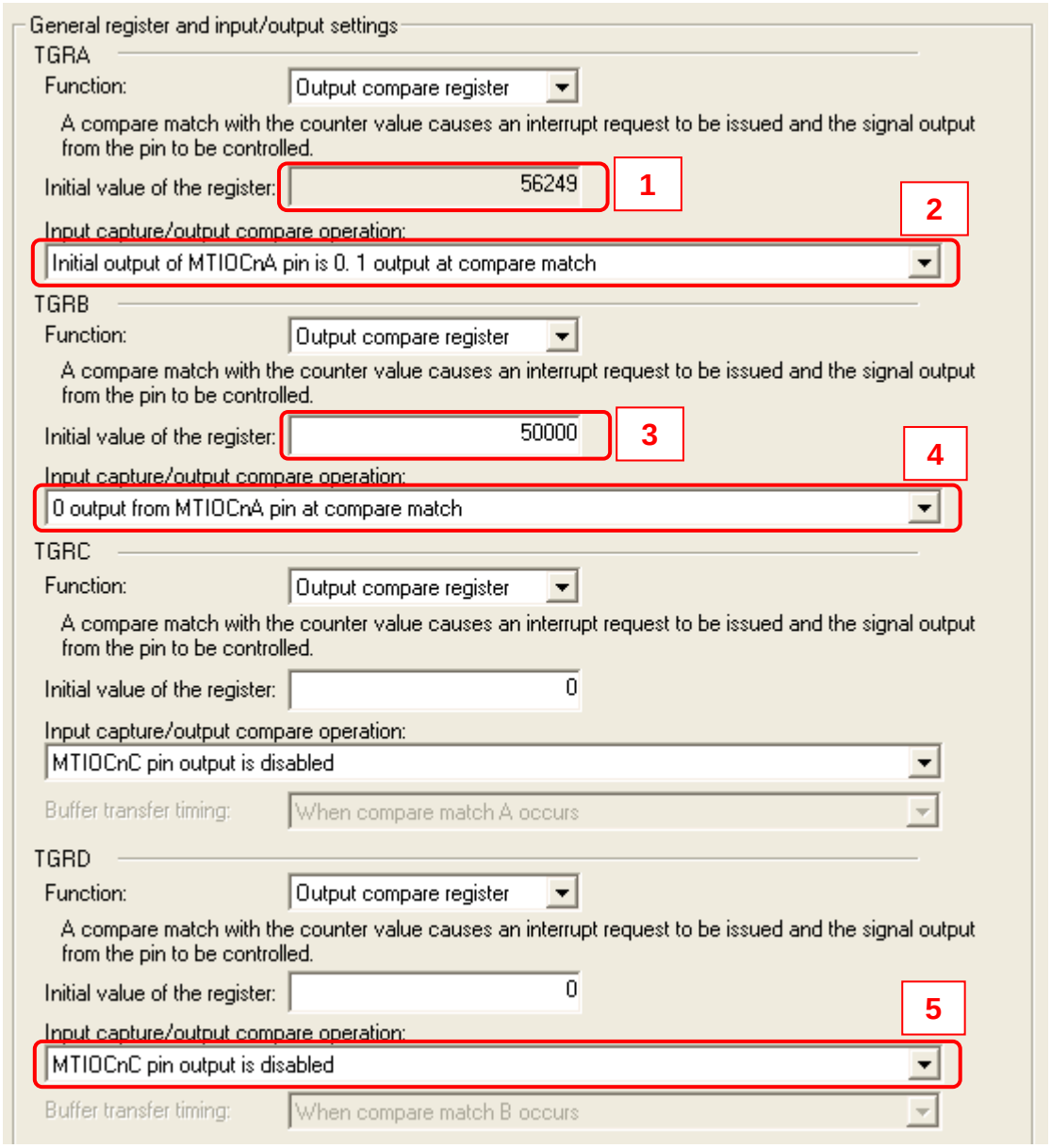
1. Select "TGRA register compare match" for a counter clearing source.
2. Select "Internal clock (PCLK/64)" for a count source.
3. Set timer operation period to "300 msec".



(6) MTU2 setting-4 **PDG**

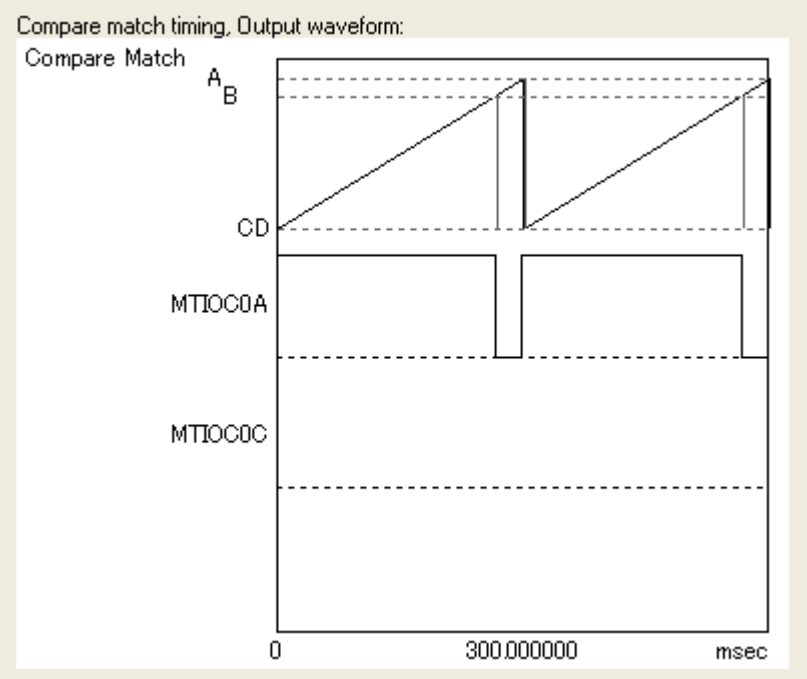
General register setting is as follows.

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 is 0, 1 output at compare match" for TGRA output compare operation.
3. Set TGRB initial value to "50000".
4. Select "0 output from MTIOCnA at compare match" for TGRB output compare operation.
5. The MTIOCnC output is not used in this tutorial. Select "MTIOCnC pin output is disabled" for TGRD output compare operation.



(7) MTU2 setting-5 **PDG**

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



(8) Generating source files **PDG**

1. To generate source files, click  on the tool bar. For details on generating source files, refer to section 6.1 (8), Generating source files.

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

Start the HEW and make RX62N workspace. For details on making HEW project, refer to section 6.1 (9), 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 6.1 (10), Adding the generated source files to the HEW projec.

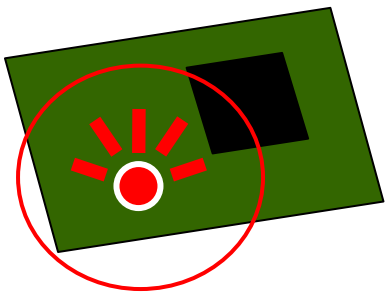
(11) Making the program on HEW **HEW**

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

```
//Include "R_PG_<PDG project name>.h"
#include "R_PG_rx62n_demo2.h"

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

      //Set up MTU2 Channel 0
      R_PG_Timer_Set_MTU_U0_C0();

      //Start the count of MTU2 Channel 0
      R_PG_Timer_StartCount_MTU_U0_C0();

      while(1);
}
```

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

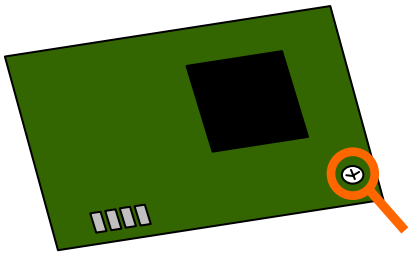
Execute the program and see the LED blinking on RSK+ board. For details on connecting to the emulator, building the program, and executing the program, refer to section 6.1 (12), connecting to the emulator, building the program and executing.



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

6.3 Continuously scanning on10-Bit A/D converter (ADa)

In RX62N RSK+ board, the potentiometer is connected to AN0 analog input. In this tutorial, the 10-Bit A/D converter (ADa) will be set up to execute A/D conversion continuously. And the result of A/D conversion will be monitored on HEW.



Potentiometer

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

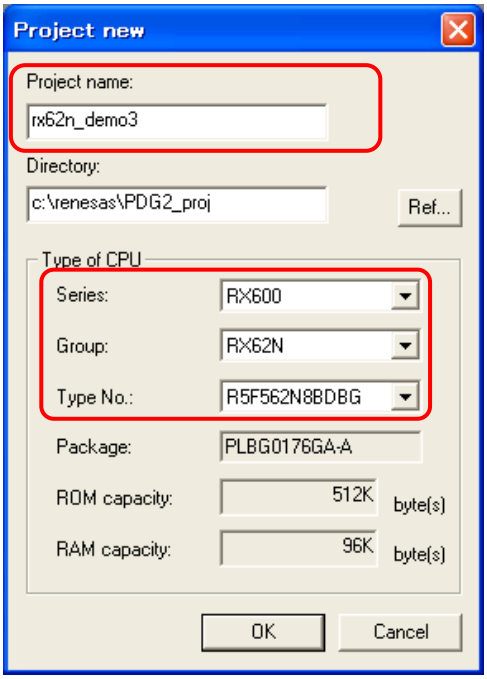
(1) Making the PDG project

PDG

Make the new PDG project “rx62n_demo3”. For details on how to make the new PDG project, refer to section 6.1 (1), Making the PDG project.
Set the CPU type as follows.

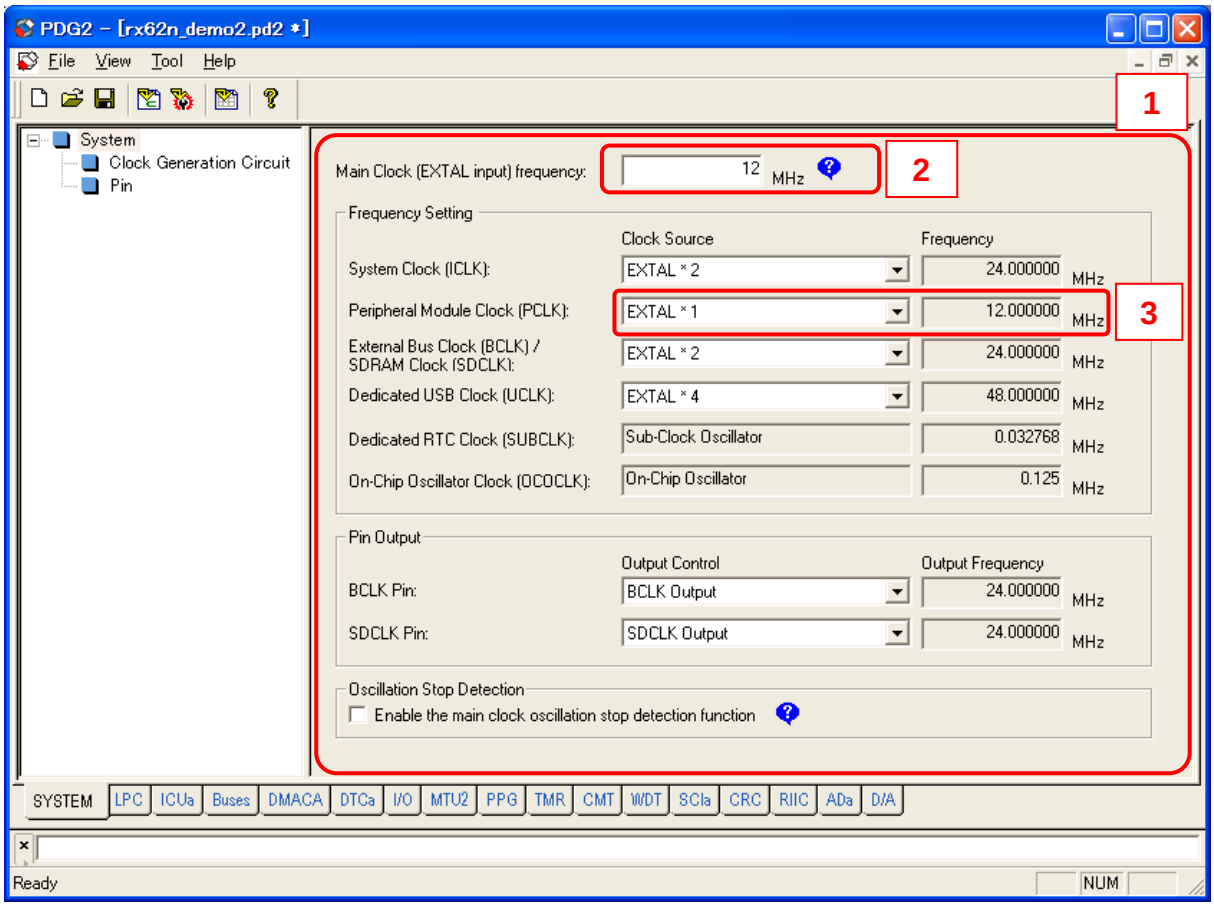
Series : RX600
Group : RX62N
Type : R5F562N8BDBG

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



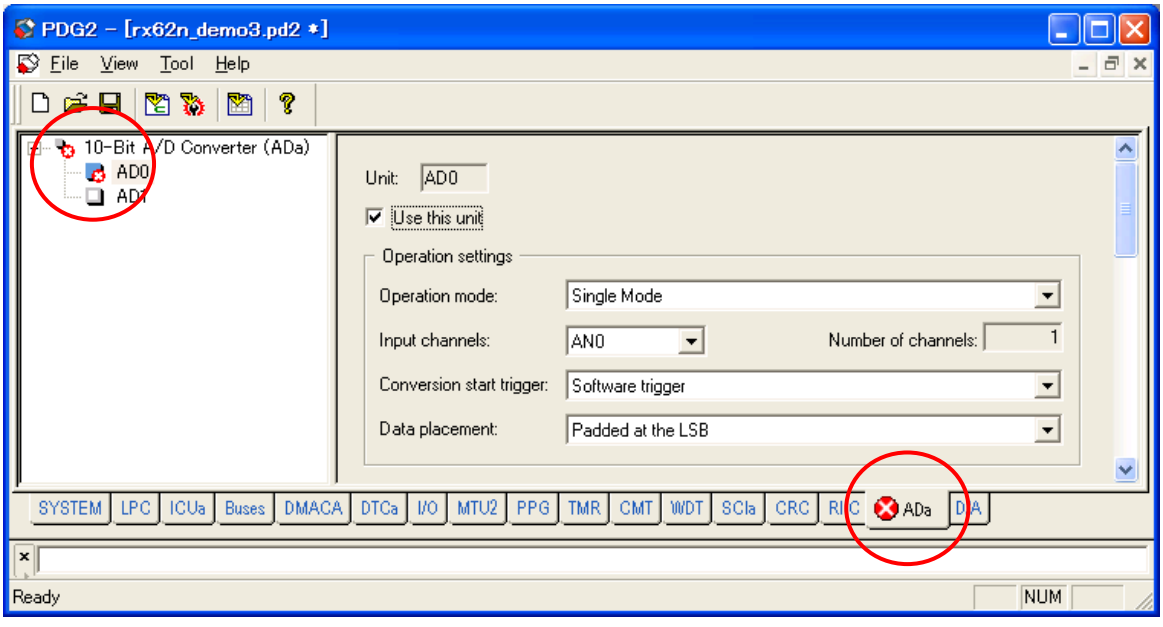
(2) Clock setting **PDG**

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 6.1 (2), Initial state.
2. External clock frequency of the RSK+ board is 12 MHz. Input “12” into the edit box.
3. PCLK is used in 12MHz. Select the multiplication "EXTAL x 1" to set the PCLK to 12MHz



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

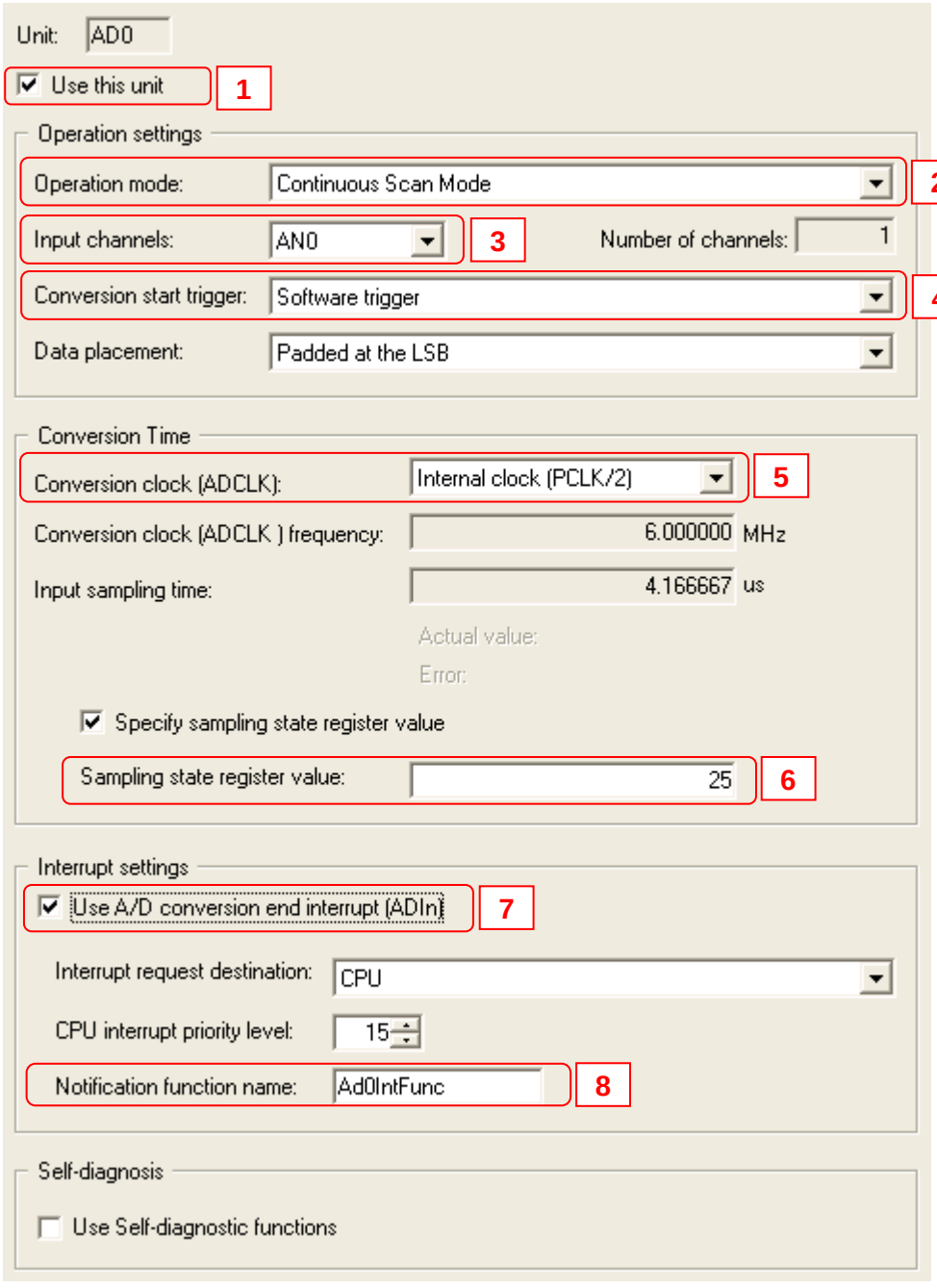
Select “ADa” tab and click AD0 on tree view.



(4) A/D converter setting-2 PDG

Make the following setting for AD0.

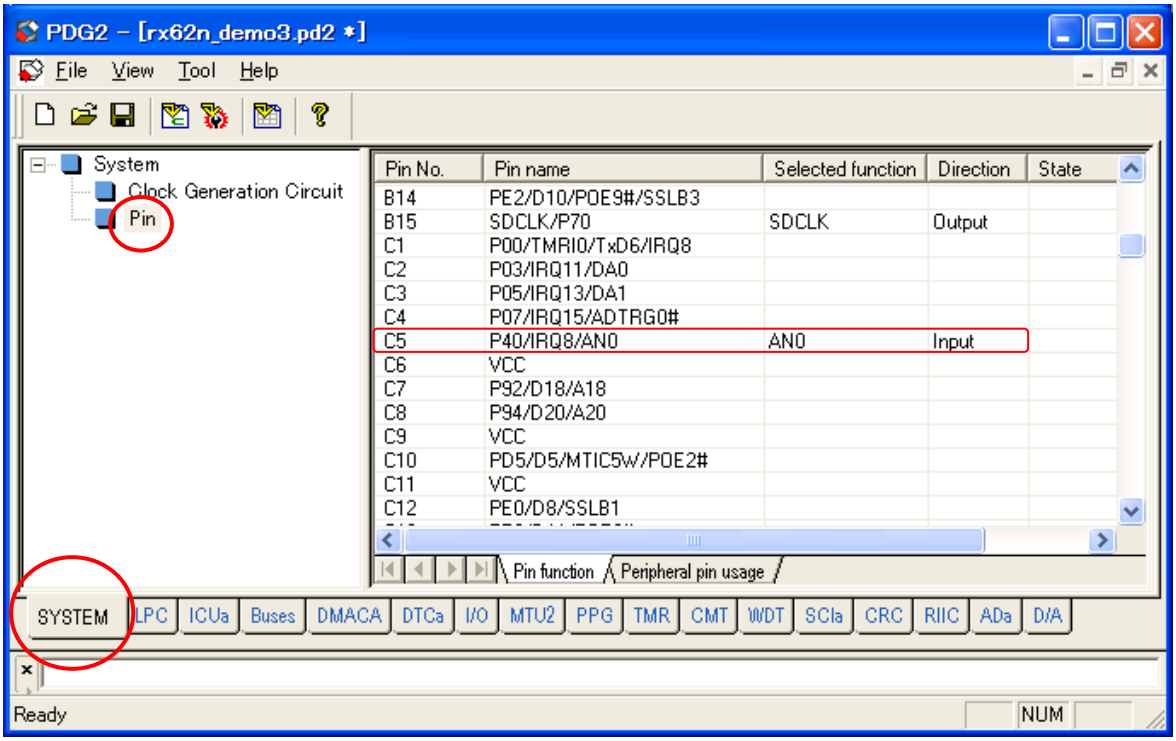
1. Check "Use this unit".
2. Select "Continuous scan mode" for the operation mode.
3. Select "Software trigger" for the conversion start trigger.
4. Select "Internal clock (PCLK/2)" for the conversion clock.
5. Set the sampling state register value to 25.
6. Set A/D conversion end interrupt notification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



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

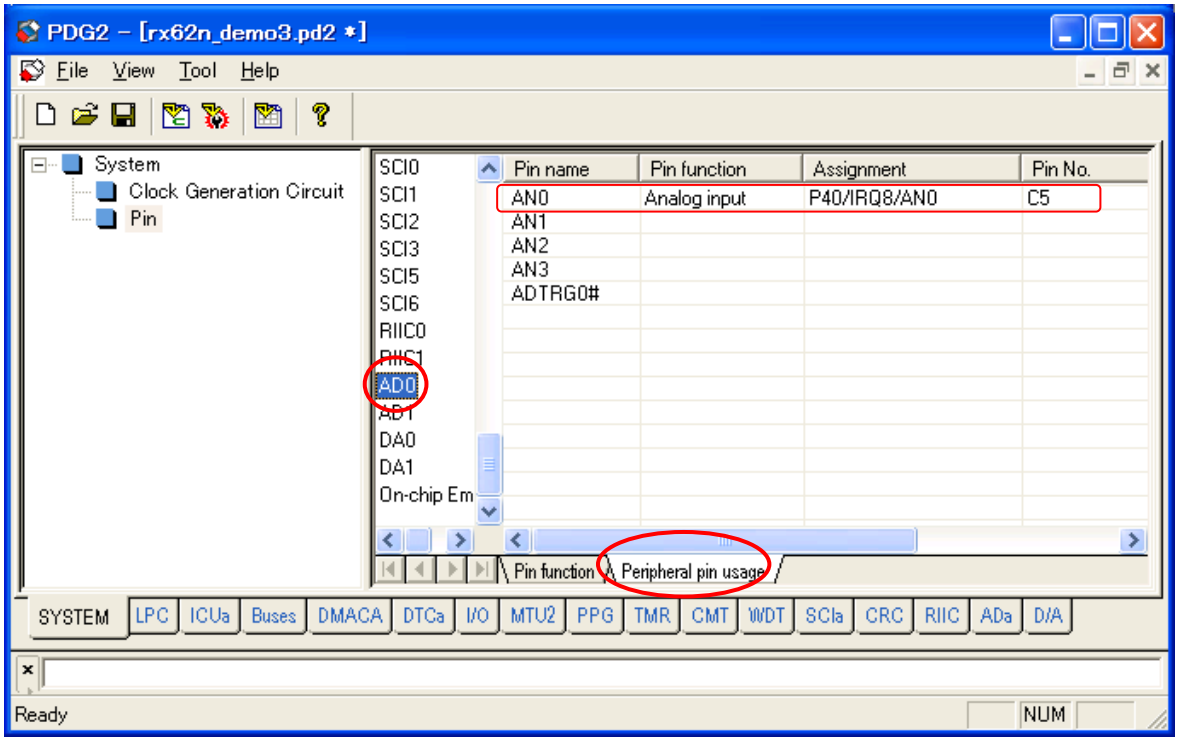
- It is possible to check the usage of pins on the pin function windows

1. After setting up the AD0, select “SYSTEM” tab and click “Pin” on the tree view.
2. On the Pin function window, you can see that No.C5 pin is used as AN0.



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

Select Peripheral pin usage sheet and click AD0 to check the usage of AN0 pin.



(6) Generating source files **PDG**

To generate source files, click  on the tool bar. For details on generating source files, refer to section 6.1 (8), Generating source files.

(7) Preparing the HEW project **HEW**

Start the HEW and make RX62N workspace. For details on making HEW project, refer to section 6.1 (9), Preparing the HEW project.

(8) 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 6.1 (10), Adding the generated source files to the HEW projec.

(9) 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_rx62n_demo3.h"
void main(void)
{
	//Set up the clock
	R_PG_Clock_Set();
	//Set up AD0
	R_PG_ADC_10_Set_AD0();
	//Start A/D conversion
	R_PG_ADC_10_StartConversionSW_AD0();
	while(1);
}
// Variable to store the result
uint16_t result;
// AD0 conversion end interrupt notification function
void Ad0IntFunc(void)
{
	// Get the result of conversion
	R_PG_ADC_10_GetResult_AD0(&result);
}
```

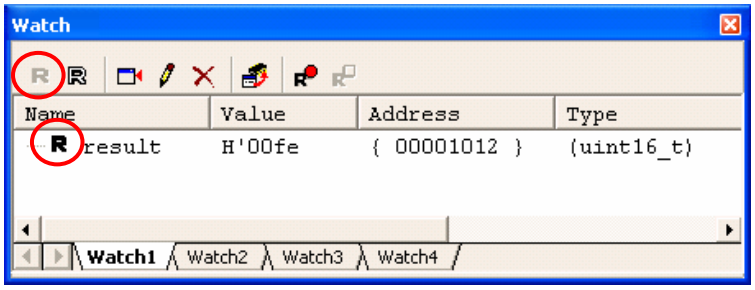
(10) Connecting to the emulator, building the program and downloading **HEW**

Buid the program and download it. For details on connecting to the emulator, building the program, and downloading refer to section 6.1 (12), connecting to the emulator, building the program and executing.

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

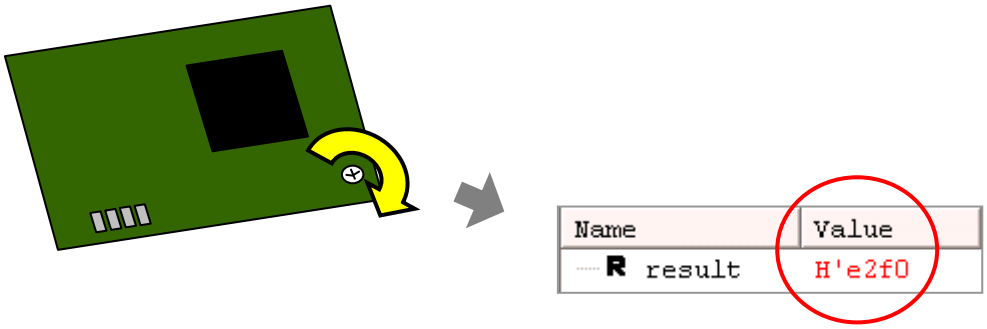
(11) Adding the variable of A/D conversion result to the watch window **HEW**

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



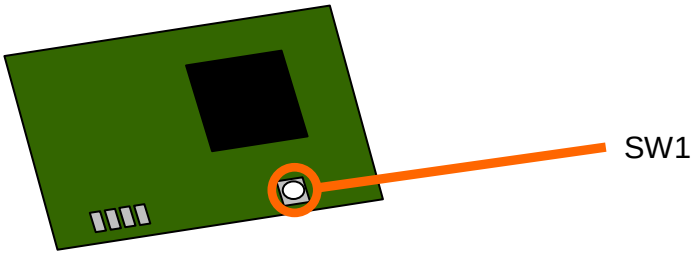
(12) Executing the program and monitoring the A/D conversion result **HEW**

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



6.4 Triggering DTC by IRQ

In RX62N RSK+ board, switch 1 (SW1) is connected to IRQ8. In this tutorial, the data transfer controller (DTCa) and IRQ8 will be set up and DTC transfer triggered by IRQ8 will be performed.



Note : If there is a switch that enables/disables IRQ8 on the RSK+ board, enable it.

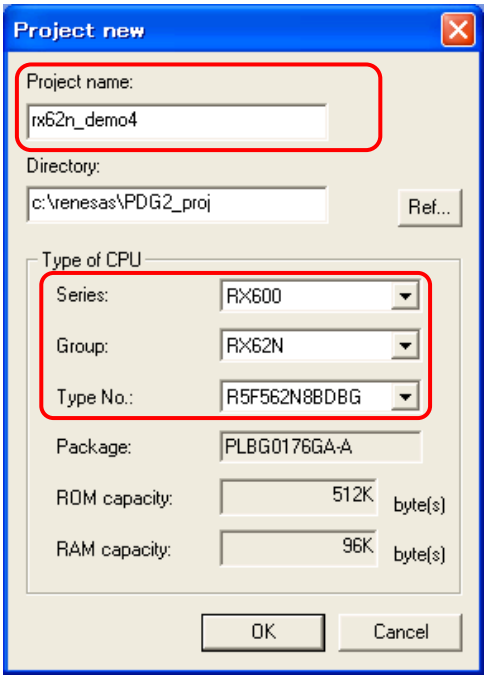
(1) Making the PDG project

PDG

Make the new PDG project “rx62n_demo4”. For details on how to make the new PDG project, refer to section 6.1 (1), Making the PDG project.
Set the CPU type as follows.

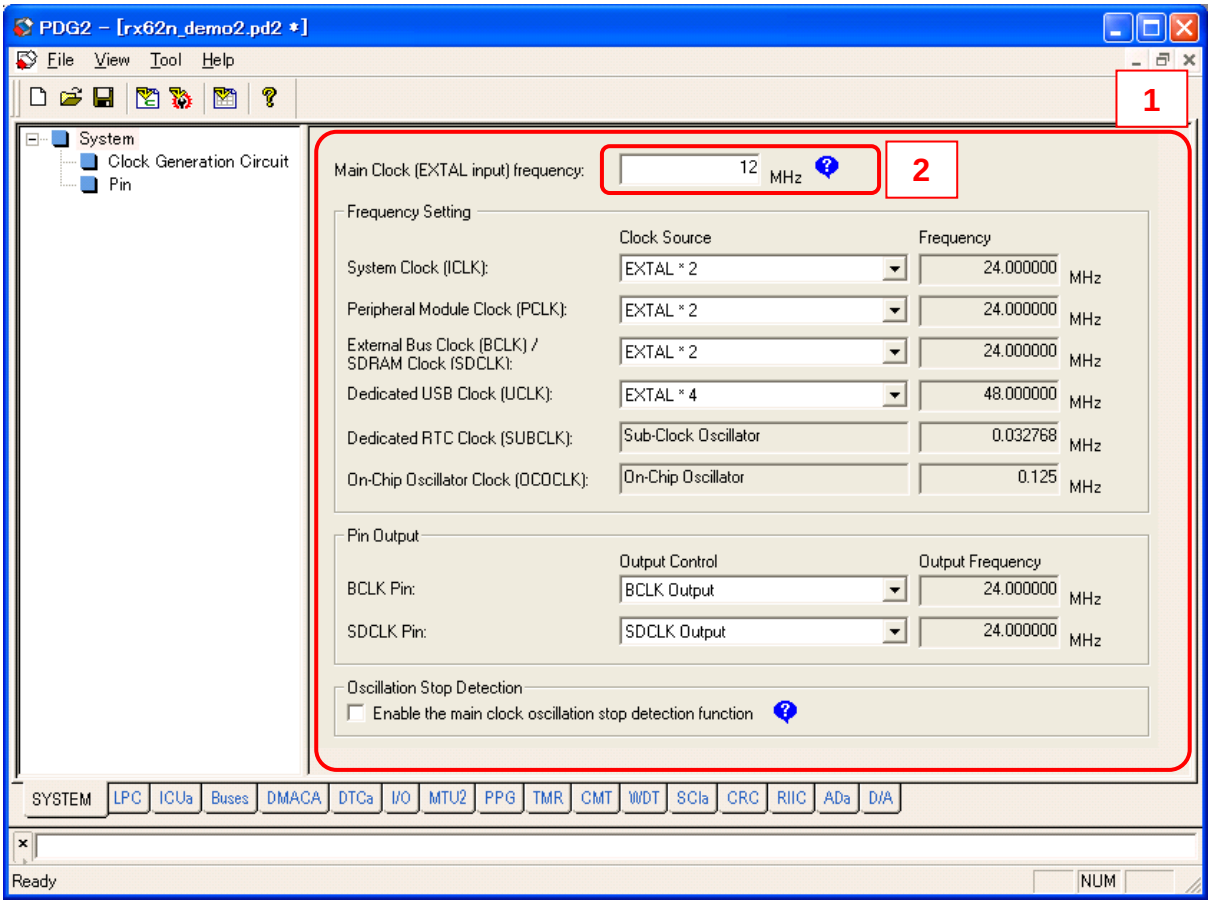
Series : RX600
Group : RX62N
Type : R5F562N8BDBG

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



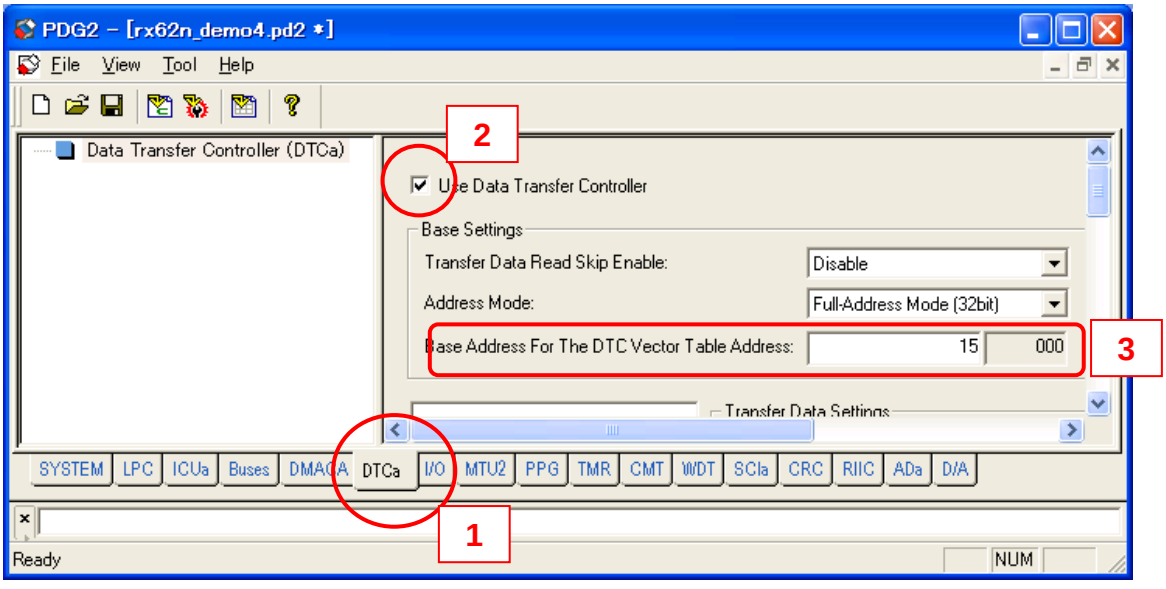
(2) Clock setting **PDG**

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 6.1 (2), Initial state.
2. External clock frequency of the RSK+ board is 12 MHz. Input “12” into the edit box.



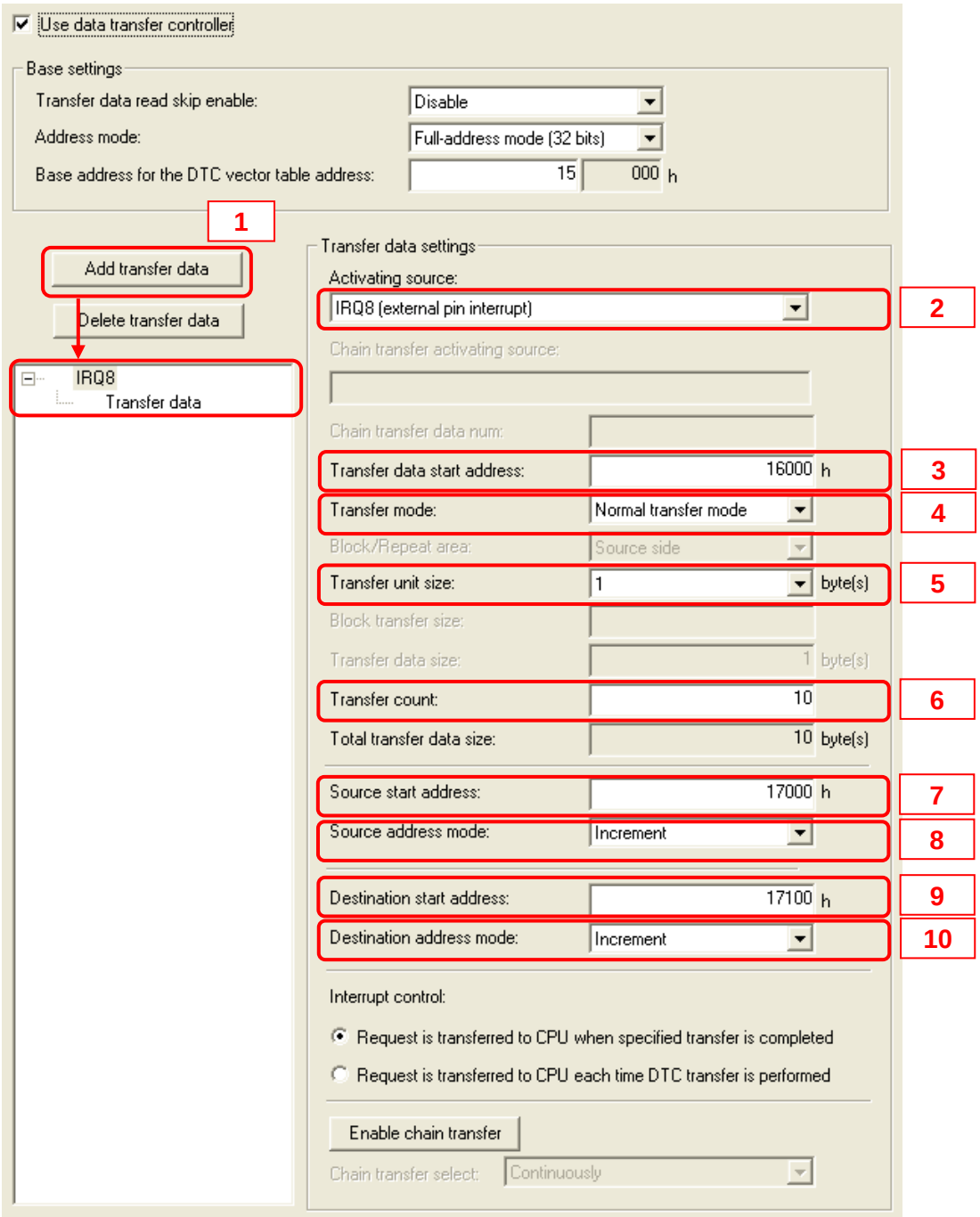
(3) DTC setting-1 **PDG**

1. Select “DTCa” tab to open the DTC setting window.
2. Check "Use data transfer controller".
3. The DTC vector table will be allocated from 15000. Input “15” into the edit box.



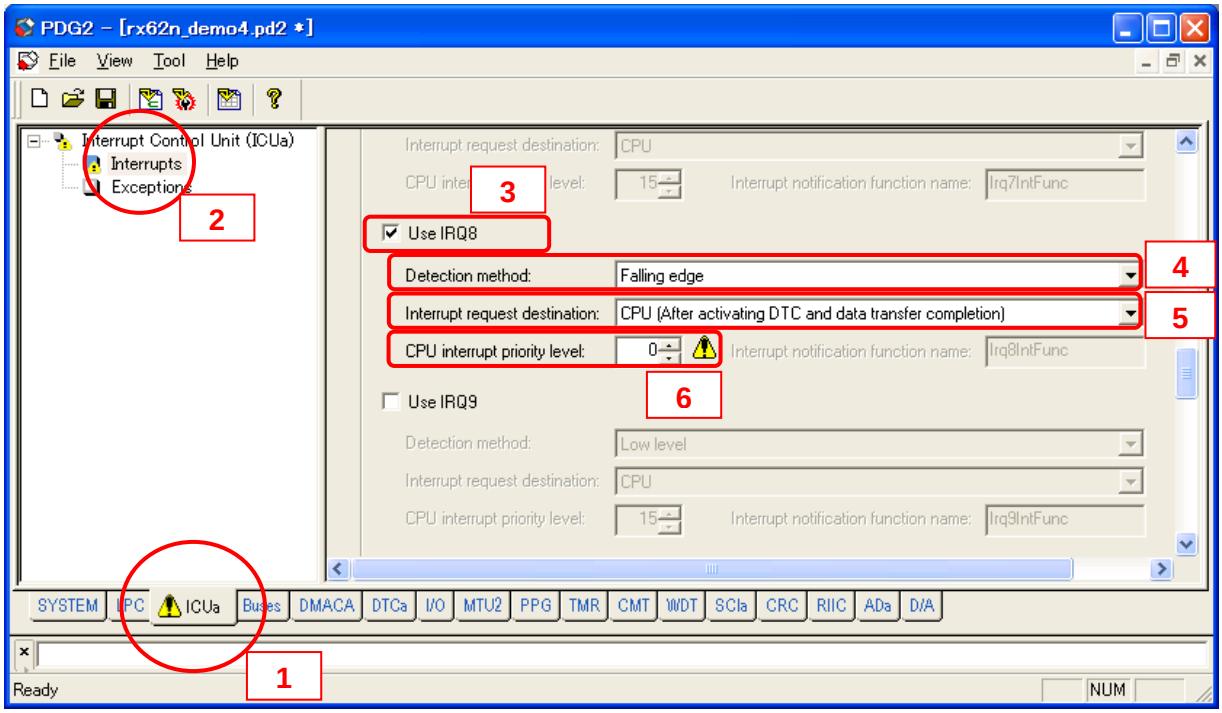
(4) DTC setting-2 **PDG**

1. Click [Add transfer data] to add the transfer data.
2. Select “IRQ8 (external pin interrupt)” for the trigger source.
3. Set the transfer data start address to “16000”.
4. Select “Normal transfer mode” for the transfer mode.
5. Set the transfer unit size to “1”.
6. Set transfer count to “10”.
7. Set the transfer source start address to “17000”.
8. Select “Increment” for thetransfer source address mode.
9. Set the transfer destination start address to “17100”.
10. Select “Increment” for thetransfer destination address mode.



(5) IRQ setting **PDG**

1. Select “ICUa” tab to open the DTC setting window.
2. Click “Interrupt” on the tree view.
3. Check "Use IRRQ8".
4. Select “Falling edge” fro te detection method of IRQ8.
5. Select “CPU (After activating DTC and data transfer completion)”
6. CPU interrupt will not be used then set the CPU interrupt priority level to “0”.



(6) Generating source files **PDG**

To generate source files, click  on the tool bar. For details on generating source files, refer to section 6.1 (8), Generating source files.

(7) Preparing the HEW project **HEW**

Start the HEW and make RX62N workspace. For details on making HEW project, refer to section 6.1 (9), Preparing the HEW project.

(8) 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 6.1 (10), Adding the generated source files to the HEW projec.

(9) 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_rx62n_demo4.h"

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

//DTC transfer data storage area (IRQ8)
#pragma address dtc_transfer_data_IRQ0 = 0x00016000
uint32_t dtc_transfer_data_IRQ0 [2];

//Transfer source
#pragma address dtc_src_data = 0x00017000
uint8_t dtc_src_data [10] = "ABCDEFGHIJ";

//Transfer destination
#pragma address dtc_dest_data = 0x00017100
uint8_t dtc_dest_data [10];

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

      // Set up the clock
      R_PG_Clock_Set();

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

      // Set up the DTC (transfer data of IRQ8)
      R_PG_DTC_Set_IRQ8();

      // Set up IRQ8
      R_PG_ExtInterrupt_Set_IRQ8();

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

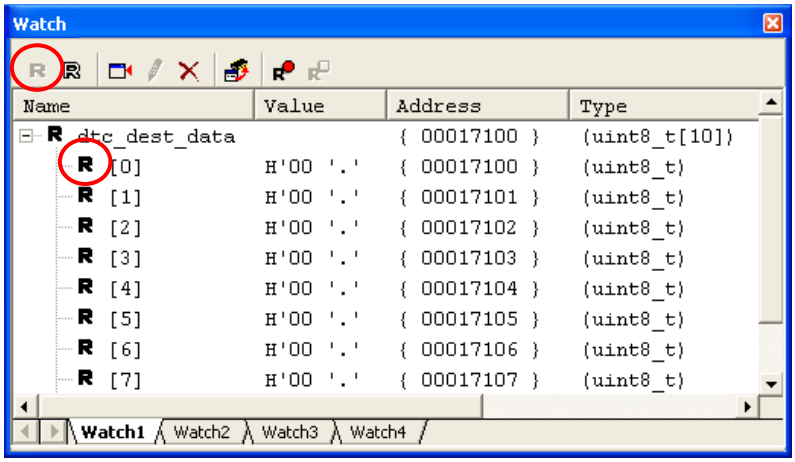
(10) Connecting to the emulator, building the program and downloading **HEW**

Buid the program and download it. For details on connecting to the emulator, building the program, and downloading refer to section 6.1 (12), connecting to the emulator, building the program and executing.

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

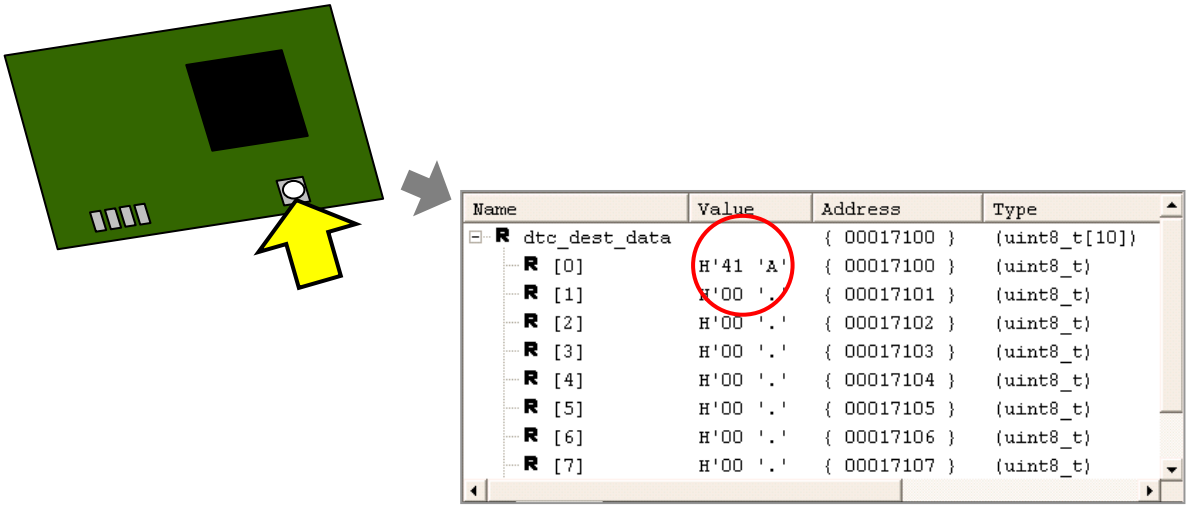
(11) Adding the variable of the transfer destination **HEW**

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



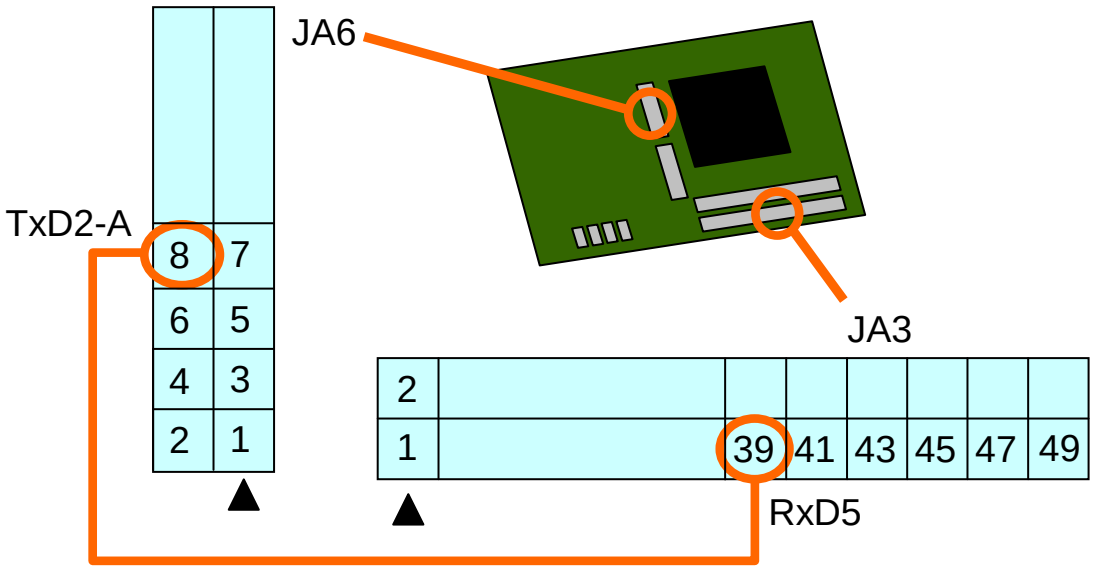
(12) Executing the program and monitoring the result of the transfer **HEW**

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



6.5 Data transfer between SCIa channels 2 and 5

In this tutorial, SCIa channel 2 and 5 will be set up to transfer data in asynchronous mode. Connect the transmission pin of channel 2 (TxD2-A) and the reception pin of channel 5 (Rxd5) on the RSK+ board as follows.



Note : If there are switches that enables/disables TxD2-A and RxD5 on the RSK+ board, enable it.

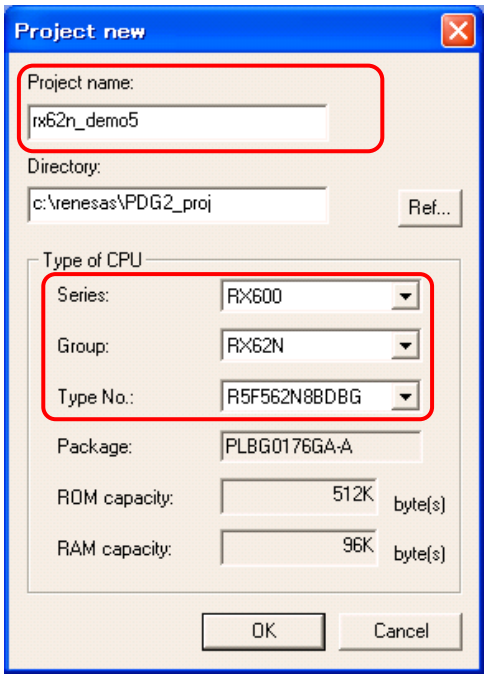
(1) Making the PDG project

PDG

Make the new PDG project “rx62n_demo5”. For details on how to make the new PDG project, refer to section 6.1 (1), Making the PDG project.
Set the CPU type as follows.

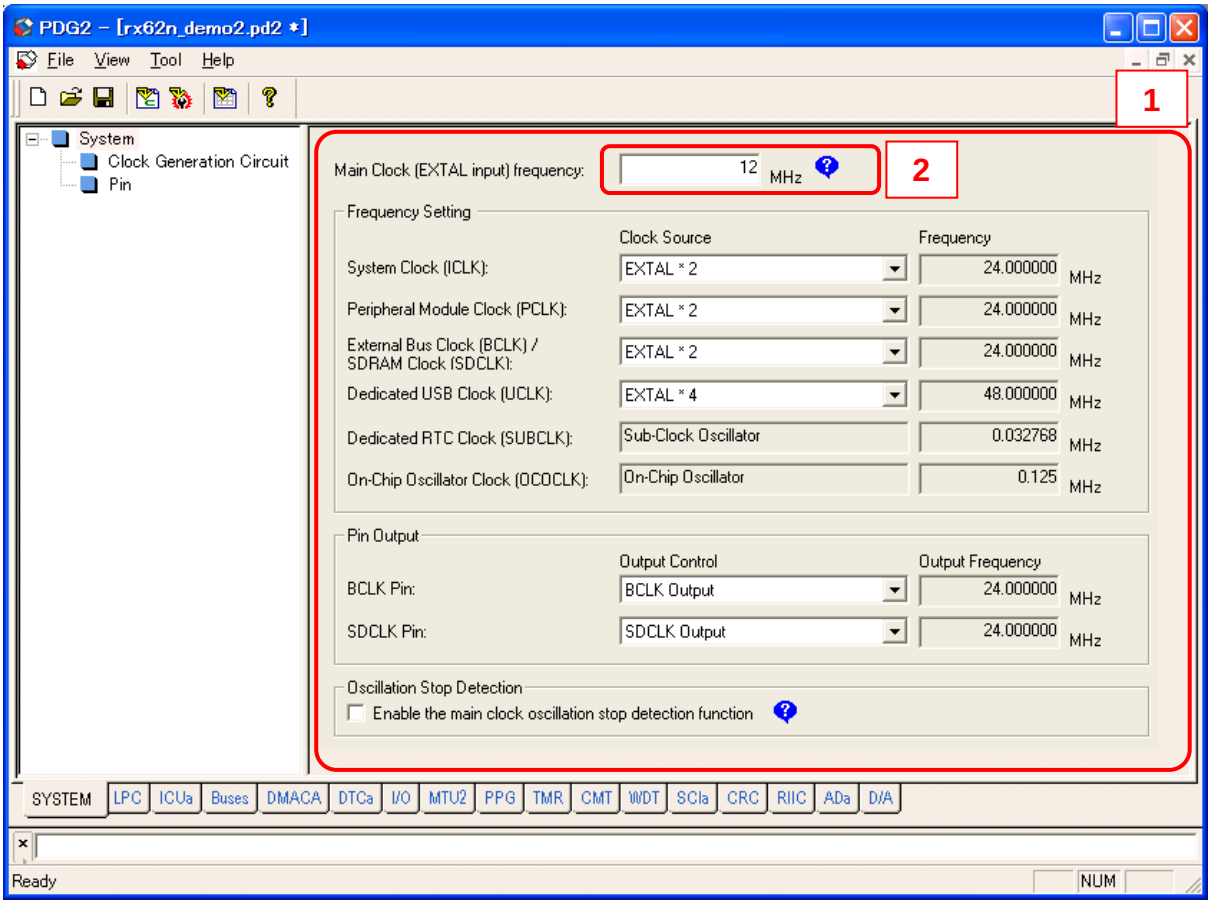
Series : RX600
Group : RX62N
Type : R5F562N8BDBG

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



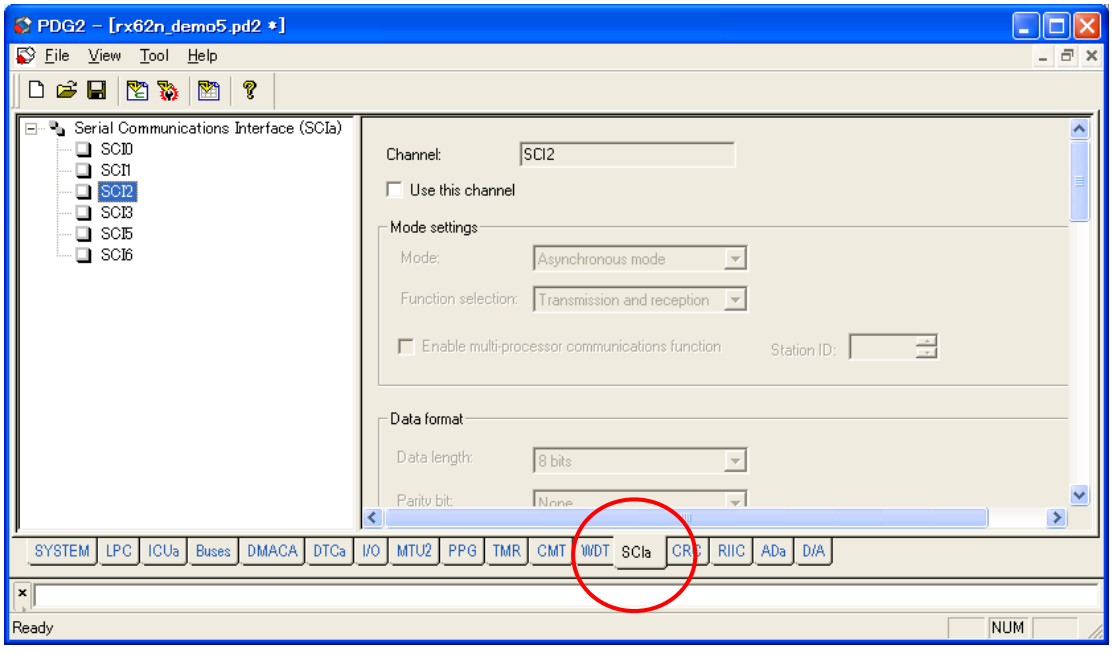
(2) Clock setting **PDG**

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 6.1 (2), Initial state.
2. External clock frequency of the RSK+ board is 12 MHz. Input “12” into the edit box.



(3) SCIa setting **PDG**

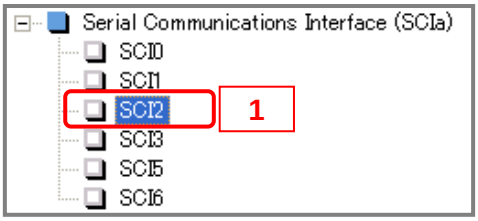
Select “SCIa” tab to open the SCIa setting window.



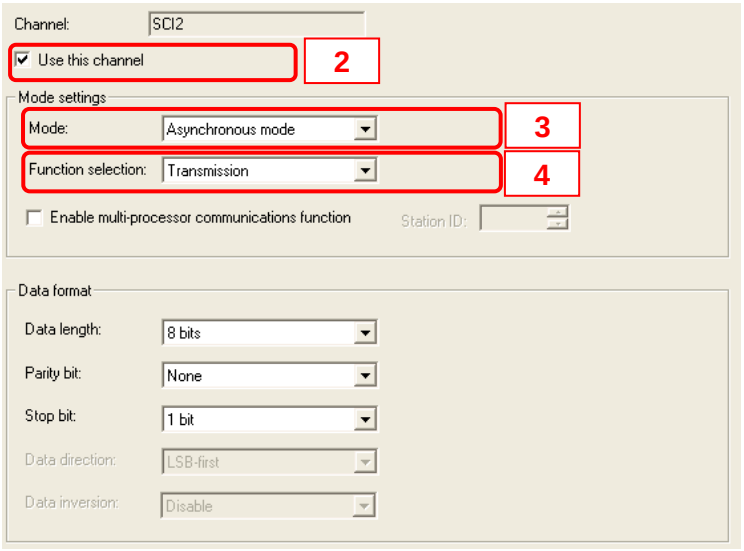
(4) SCI2 (transmitter) setting PDG

Make the setting for SCI2 as follows.

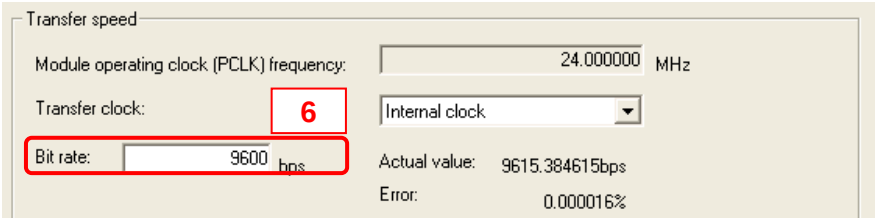
1. Select SCI2 on the tree view.



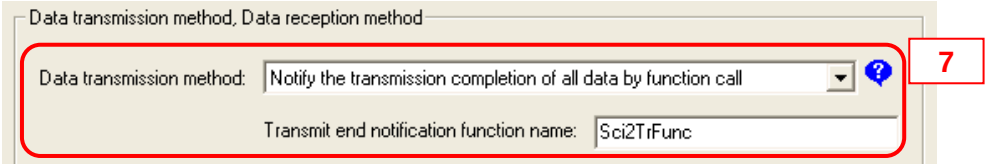
2. Check “Use this channel”.
3. Select “Asynchronous mode”.
4. Select “Transmission” for the function.
5. Leave the data format settings at the default.



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



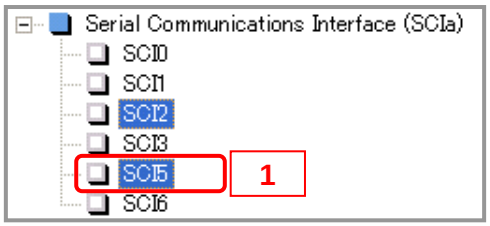
7. Select “Notify the transmission completion of all data by function call” for the data transmission method.



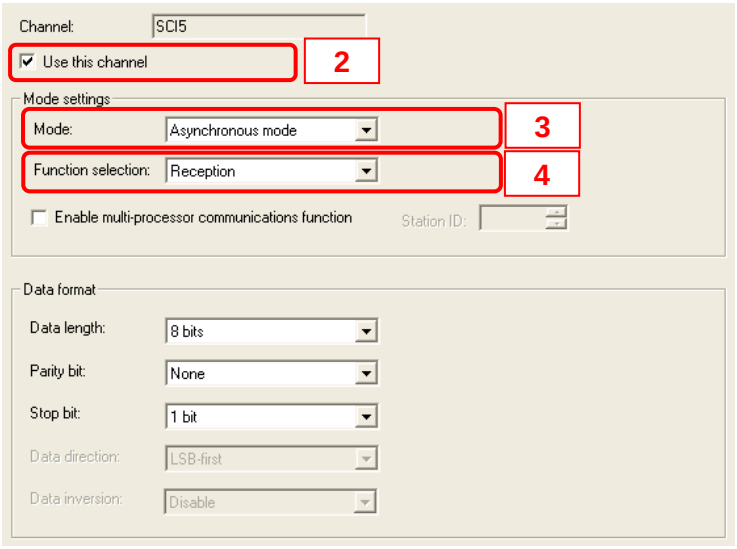
(5) SCI5 (receptor) setting PDG

Make the setting for SCI5 as follows.

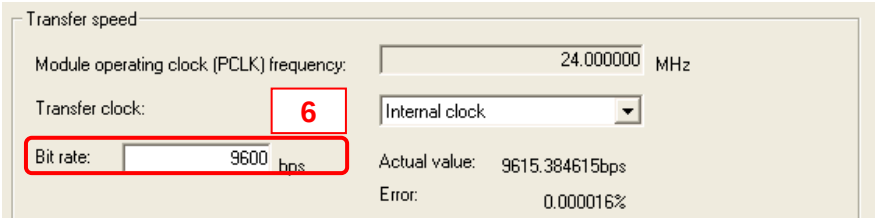
1. Select SCI5 on the tree view.



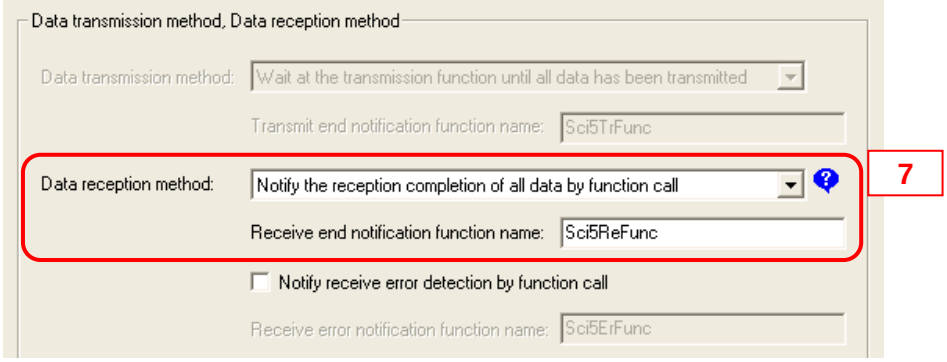
2. Check “Use this channel”.
3. Select “Asynchronous mode”.
4. Select “Reception” for the function.
5. Leave the data format settings at the default.



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



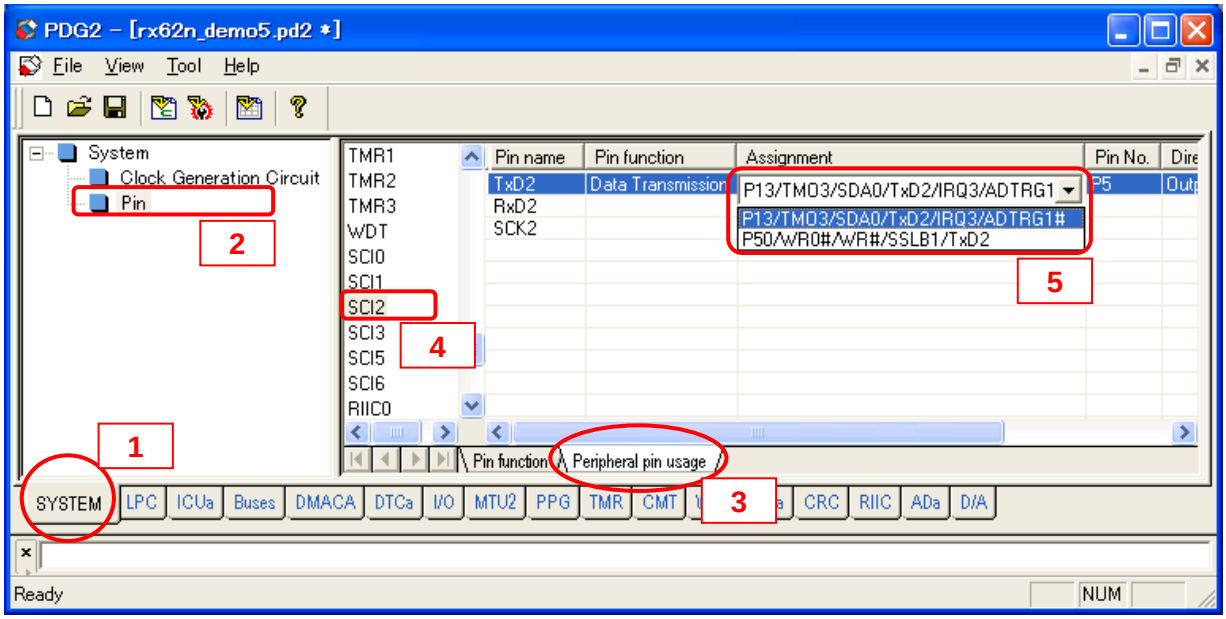
7. Select “Notify the reception completion of all data by function call” for the data reception method.



(6) Pin setting **PDG**

The TxD2 can be assigned to TxD2-A (P13) or TxD2-B (P50). Select the pin function assignment as follows.

1. Select “SYSTEM” tab.
2. Select “Pin” on tree view.
3. Select “Peripheral pin usage” tab.
4. Select “SCI2” from the peripheral module list.
5. When the mouse pointer is placed on “Assignment” column of TxD2 line, a dropdown button is displayed. Select “P13/TMO3/SDA0/TxD2/IRQ3/ADTRG1#” from the dropdown list.



(7) Generating source files **PDG**

To generate source files, click  n the tool bar. For details on generating source files, refer to section 6.1 (8), Generating source files.

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

Start the HEW and make RX62N workspace. For details on making HEW project, refer to section 6.1 (9), 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 6.1 (10), Adding the generated source files to the HEW projec.

(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_rx62n_demo5.h"

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

//SCI5 reception data storage area
uint8_t re_data[10] = "----------";

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

      // Set up the SCI2
      R_PG_SCI_Set_C2();

      // Set up the SCI5
      R_PG_SCI_Set_C5();

      // Start SCI5 reception (number of data : 10)
      R_PG_SCI_StartReceiving_C5( re_data, 10 );

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

      while(1);
}

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

//SCI5 reception end notification function
void Sci5ReFunc(void)
{
      //Stop SCI5 communication
      R_PG_SCI_StopCommunication_C5();
}
```

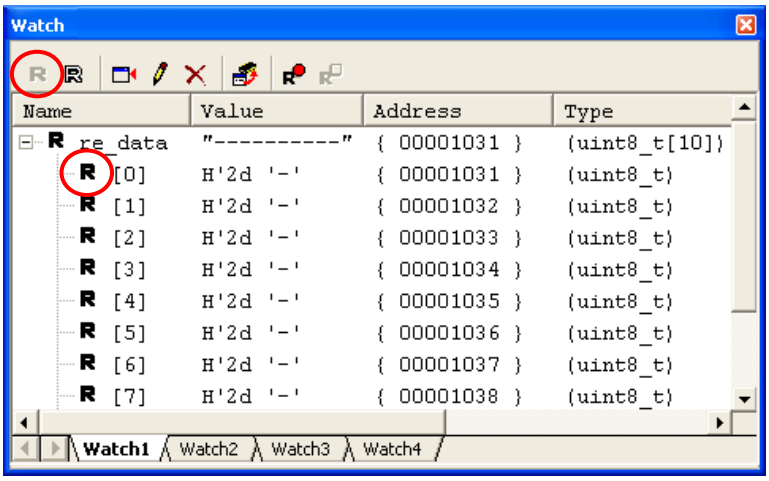
(11) Connecting to the emulator, building the program and downloading **HEW**

Buid the program and download it. For details on connecting to the emulator, building the program, and downloading refer to section 6.1 (12), connecting to the emulator, building the program and executing.

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

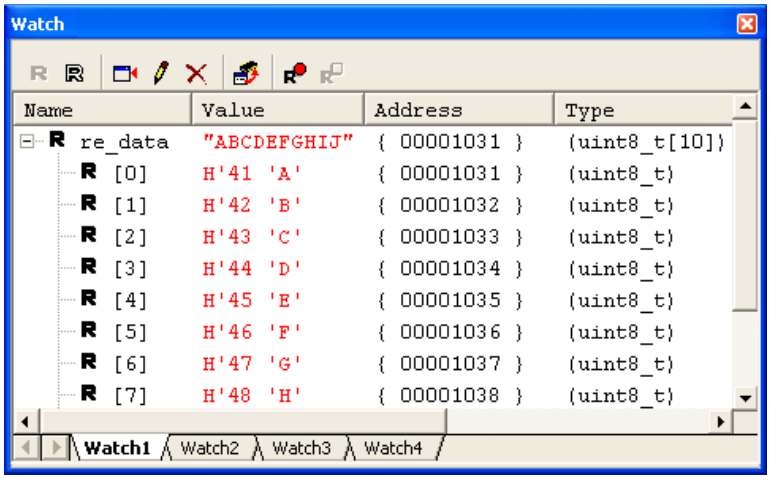
(12) Adding the variable of the reception data **HEW**

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



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

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



Appendix 1. Pin Functions for which the Allocation Can be Changed

Table a-1.1 176-pin LFBGA (the Upper Row of Each Pair is the Default Selection)

Peripheral module	Pin function	Selection of assignment	Pin No.
ICU (External Interrupts)	IRQ0	P30/USB1_DRPD/MTIOC4B/TMRI3/PO8/MISOB/RxD1/IRQ0	L4
		P10/USB1_DPUPE/MTIC5W/TMRI3/IRQ0	N7
	IRQ1	P31/USB1_DPRPD/MTIOC4D/TMCI2/PO9/SSLB0/IRQ1	K2
		P11/USB1_VBUSEN/MTIC5V/TMCI3/SCK2/IRQ1	M5
	IRQ2	P32/MTIOC0C/PO10/RTCOUT/CTX0/TxD6/IRQ2	J2
		P12/MTIC5U/TMCI1/SCL0/RxD2/IRQ2	R3
	IRQ3	P33/MTIOC0D/PO11/CRX0/RxD6/IRQ3	K1
		P13/TMO3/SDA0/TxD2/IRQ3/ADTRG1#	P5
	IRQ4	P34/MTIOC0A/TMCI3/PO12/SCK6/IRQ4	J4
		P14/USB0_OVRCURA/USB0_DPUPE/TMRI2/IRQ4	P4
	IRQ5	PE5/D13/RSPCKB/IRQ5	C14
		P15/USB1_OVRCURA/USB1_DPUPE/MTIOC0B/TMCI2/PO13/ SCK3/IRQ5	N5
	IRQ6	PE6/D14/MOSIB/IRQ6	C15
		P16/USB0_VBUS/USB0_OVRCURB/USB0_VBUSEN/MTIOC3C/ TMO2/PO14/RxD3/IRQ6	P3
	IRQ7	PE7/D15/MISOB/IRQ7	D14
		P17/USB1_VBUS/USB1_OVRCURB/USB1_VBUSEN/MTIOC3A/ PO15/TxD3/IRQ7	N4
	IRQ8	P00/TMRI0/TxD6/IRQ8	C1
		P40/IRQ8/AN0	C5
	IRQ9	P01/TMCI0/RxD6/IRQ9	D2
		P41/IRQ9/AN1	D4
	IRQ10	P02/TMCI1/SCK6/IRQ10	B1
		P42/IRQ10/AN2	A3
	IRQ11	P03/IRQ11/DA0	C2
		P43/IRQ11/AN3	D5
	IRQ13	P05/IRQ13/DA1	C3
		P45/IRQ13/AN5	A4
	IRQ15	P07/IRQ15/ADTRG0#	C4
		P47/IRQ15/AN7	B5
Addres Bus	A16	PC0/A16/ET_ERXD3/MTCLKG/SSLA1	M12
	*1	P90/D16/A16	A6
	A17	PC1/A17/ET_RXD2/MTCLKH/SSLA2/SCK5	P14
	*1	P91/D17/A17	B6
	A18	PC2/A18/ET_RX_DV/MTCLKE/SSLA3/RxD5	N12
	*1	P92/D18/A18	C7
	A19	PC3/A19/ET_TX_ER/MTCLKF/TxD5	N11
	*1	P93/D19/A19	D7
	A20	PC4/A20/CS3#/ET_TX_CLK/MTCLKC/SSLA0	P12
	*1	P94/D20/A20	C8
	A21	PC5/A21/CS2#/WAIT#/ET_ETXD2/MTIC11W/MTCLKD/RSPCKA	N10
	*1	P95/D21/A21	D8
	A22	PC6/A22/CS1#/ET_ETXD3/MTIC11V/MTCLKA/MOSIA	M10
	*1	P96/D22/A22	B8
	A23	PC7/A23/CS0#/ET_COL/MTIC11U/MTCLKB/MISOA	R12
	*1	P97/D23/A23	B9

Bus control	CS0#	P60/CS0#	B11
		PC7/A23/CS0#/ET_COL/MTIC11U/MTCLKB/MISOA	R12
	CS1#	P61/CS1#/SDCS#	A13
		P71/CS1#/ET_MDIO	K13
		PC6/A22/CS1#/ET_ETXD3/MTIC11V/MTCLKA/MOSIA	M10
	CS2#	P62/CS2#/RAS#	B12
		P72/CS2#/ET_MDC	K14
		PC5/A21/CS2#/WAIT#/ET_ETXD2/MTIC11W/MTCLKD/RSPCKA	N10
	CS3#	P63/CS3#/CAS#	A14
		P73/CS3#/ET_WOL	N14
		PC4/A20/CS3#/ET_TX_CLK/MTCLKC/SSLA0	P12
	CS4#	P64/CS4#/WE#	B13
		P74/CS4#/ET_ERXD1/RMII_RXD1	N13
		P24/CS4#/EDREQ1/USB0_VBUSEN/MTIOC4A/MTCLKA/TMRI1/ PO4/SCK3	P1
	CS5#	P65/CS5#/CKE	D15
		P75/CS5#/ET_ERXD0/RMII_RXD0	R15
		P25/CS5#/EDACK1/USB0_DPRPD/MTIOC4C/MTCLKB/PO5/ RxD3/ADTRG0#	M2
	CS6#	P66/CS6#/DQM0	E14
		P76/CS6#/ET_RX_CLK/REF50CK	P13
		P26/CS6#/USB1_ID/MTIOC2A/TMO1/PO6/MOSIB/TxD1	N1
	CS7#	P67/CS7#/DQM1	E15
		P77/CS7#/ET_RX_ER/RMII_RX_ER	R14
		P27/CS7#/USB1_EXICEN/MTIOC2B/PO7/RSPCKB/SCK1	L2
	WAIT#	P57/WAIT#/WR3#/BC3#/EDREQ1	N6
		P55/WAIT#/EDREQ0/ET_EXOUT/MTIOC4D	M6
		PC5/A21/CS2#/WAIT#/ET_ETXD2/MTIC11W/MTCLKD/RSPCKA	N10
		P51/WR1#/BC1#/WAIT#/SSLB2/SCK2	M8
MTU0-5	MTCLKA	P24/CS4#/EDREQ1/USB0_VBUSEN/MTIOC4A/MTCLKA/TMRI1/ PO4/SCK3	P1
	*2	PC6/A22/CS1#/ET_ETXD3/MTIC11V/MTCLKA/MOSIA	M10
	MTCLKB	P25/CS5#/EDACK1/USB0_DPRPD/MTIOC4C/MTCLKB/PO5/ RxD3/ADTRG0#	M2
	*2	PC7/A23/CS0#/ET_COL/MTIC11U/MTCLKB/MISOA	R12
	MTCLKC	P22/EDREQ0/USB0_DRPD/MTIOC3B/MTCLKC/TMO0/PO2/SCK0	M3
	*2	PC4/A20/CS3#/ET_TX_CLK/MTCLKC/SSLA0	P12
	MTCLKD	P23/EDACK0/USB0_DPUPE/MTIOC3D/MTCLKD/PO3/TxD3	N2
	*2	PC5/A21/CS2#/WAIT#/ET_ETXD2/MTIC11W/MTCLKD/RSPCKA	N10
MTU6-11	MTCLKE	PC2/A18/ET_RX_DV/MTCLKE/SSLA3/RxD5	N12
	*3	PB4/A12/MTIOC10A/MTCLKE/PO28	L13
	MTCLKF	PC3/A19/ET_TX_ER/MTCLKF/TxD5	N11
	*3	PB5/A13/MTIOC10C/MTCLKF/PO29	N15
	MTCLKG	PC0/A16/ET_ERXD3/MTCLKG/SSLA1	M12
	*3	PB2/A10/MTIOC9B/MTCLKG/PO26	M15
	MTCLKH	PC1/A17/ET_RXD2/MTCLKH/SSLA2/SCK5	P14
	*3	PB3/A11/MTIOC9D/MTCLKH/PO27	L14
MTU3	MTIOC3B	P22/EDREQ0/USB0_DRPD/MTIOC3B/MTCLKC/TMO0/PO2/SCK0	M3
	*4	P80/EDREQ0/ET_TX_EN/RMII_TXDN/MTIOC3B	R13
	MTIOC3C	P16/USB0_VBUS/USB0_OVRCURB/USB0_VBUSEN/MTIOC3C/ TMO2/PO14/RxD3/IRQ6	P3
		P56/WR2#/BC2#/EDACK1/MTIOC3C	P7
	MTIOC3D	P23/EDACK0/USB0_DPUPE/MTIOC3D/MTCLKD/PO3/TxD3	N2

	*4	P81/EDACK0/ET_ETXD0/RMII_TXD0/MTIOC3D	M11
MTU4	MTIOC4A	P24/CS4#/EDREQ1/USB0_VBUSEN/MTIOC4A/MTCLKA/TMRI1/PO4/SCK3	P1
	*5	P82/EDREQ1/ET_ETXD1/RMII_TXD1/MTIOC4A	P11
	MTIOC4B	P30/USB1_DRPD/MTIOC4B/TMRI3/PO8/MISOB/RxD1/IRQ0	L4
	*6	P54/EDACK0/ET_LINKSTA/MTIOC4B	M7
	MTIOC4C	P25/CS5#/EDACK1/USB0_DPRPD/MTIOC4C/MTCLKB/PO5/RxD3/ADTRG0#	M2
	*5	P83/EDACK1/ET_CRS/RMII_CRS_DV/MTIOC4C	R11
	MTIOC4D	P31/USB1_DPRPD/MTIOC4D/TMCI2/PO9/SSLB0/IRQ1	K2
	*6	P55/WAIT#/EDREQ0/ET_EXOUT/MTIOC4D	M6
MTU5	MTIC5U	P12/MTIC5U/TMCI1/SCL0/RxD2/IRQ2	R3
	*7	PD7/D7/MTIC5U/POE0#	A12
	MTIC5V	P11/USB1_VBUSEN/MTIC5V/TMCI3/SCK2/IRQ1	M5
	*7	PD6/D6/MTIC5V/POE1#	B10
	MTIC5W	P10/USB1_DPUPE/MTIC5W/TMRI3/IRQ0	N7
	*7	PD5/D5/MTIC5W/POE2#	C10
MTU11	MTIC11U	PC7/A23/CS0#/ET_COL/MTIC11U/MTCLKB/MISOA	R12
	*8	PD4/D4/MTIC11U/POE3#	A10
	MTIC11V	PC6/A22/CS1#/ET_ETXD3/MTIC11V/MTCLKA/MOSIA	M10
	*8	PD3/D3/MTIC11V/POE4#	A9
	MTIC11W	PC5/A21/CS2#/WAIT#/ET_ETXD2/MTIC11W/MTCLKD/RSPCKA	N10
	*8	PD2/D2/MTIC11W/POE5#	A8
TMR0	TMCI0	P01/TMCI0/RxD6/IRQ9	D2
	*9	P21/USB0_EXICEN/MTIOC1B/TMCI0/PO1/SCL1/RxD0	R1
	TMRI0	P00/TMRI0/TxD6/IRQ8	C1
	*9	P20/USB0_ID/MTIOC1A/TMRI0/PO0/SDA1/TxD0	N3
TMR1	TMCI1	P02/TMCI1/SCK6/IRQ10	B1
		P12/MTIC5U/TMCI1/SCL0/RxD2/IRQ2	R3
TMR2	TMCI2	P15/USB1_OVRCURA/USB1_DPUPE/MTIOC0B/TMCI2/PO13/SCK3/IRQ5	N5
		P31/USB1_DPRPD/MTIOC4D/TMCI2/PO9/SSLB0/IRQ1	K3
TMR3	TMCI3	P11/USB1_VBUSEN/MTIC5V/TMCI3/SCK2/IRQ1	M5
	*10	P34/MTIOC0A/TMCI3/PO12/SCK6/IRQ4	J4
	TMRI3	P10/USB1_DPUPE/MTIC5W/TMRI3/IRQ0	N7
	*10	P30/USB1_DRPD/MTIOC4B/TMRI3/PO8/MISOB/RxD1/IRQ0	L4
SCI1	TxD1	P26/CS6#/USB1_ID/MTIOC2A/TMO1/PO6/MOSIB/TxD1	N1
	*11	PF0/TxD1/TDO	K3
	RxD1	P30/USB1_DRPD/MTIOC4B/TMRI3/PO8/MISOB/RxD1/IRQ0	L4
	*11	PF2/RxD1/TDI	L1
	SCK1	P27/CS7#/USB1_EXICEN/MTIOC2B/PO7/RSPCKB/SCK1	L2
	*11	PF1/SCK1/TCK	M1
SCI2	TxD2	P13/TMO3/SDA0/TxD2/IRQ3/ADTRG1#	P5
	*12	P50/WR0#/WR#/SSLB1/TxD2	P10
	RxD2	P12/MTIC5U/TMCI1/SCL0/RxD2/IRQ2	R3
	*12	P52/RD#/SSLB3/RxD2	N7
	SCK2	P11/USB1_VBUSEN/MTIC5V/TMCI3/SCK2/IRQ1	M5
	*12	P51/WR1#/BC1#/WAIT#/SSLB2/SCK2	M8
SCI3	TxD3	P17/USB1_VBUS/USB1_OVRCURB/USB1_VBUSEN/MTIOC3A/PO15/TxD3/IRQ7	N4
	*13	P23/EDACK0/USB0_DPUPE/MTIOC3D/MTCLKD/PO3/TxD3	N2
	RxD3	P16/USB0_VBUS/USB0_OVRCURB/USB0_VBUSEN/MTIOC3C/TMO2/PO14/RxD3/IRQ6	P3

	*13	P25/CS5#/EDACK1/USB0_DPRPD/MTIOC4C/MTCLKB/PO5/ RxD3/ADTRG0#	M2
	SCK3	P15/USB1_OVRCURA/USB1_DPUPE/MTIOC0B/TMCI2/PO13/ SCK3/IRQ5	N5
	*13	P24/CS4#/EDREQ1/USB0_VBUSEN/MTIOC4A/MTCLKA/TMRI1/ PO4/SCK3	P1
SCI6	TxD6	P00/TMRI0/TxD6/IRQ8	C1
	*14	P32/MTIOC0C/PO10/RTCOUT/CTX0/TxD6/IRQ2	J2
	RxD6	P01/TMCI0/RxD6/IRQ9	D2
	*14	P33/MTIOC0D/PO11/CRX0/RxD6/IRQ3	K1
	SCK6	P02/TMCI1/SCK6/IRQ10	B1
	*14	P34/MTIOC0A/TMCI3/PO12/SCK6/IRQ4	J4
AD0, S12AD0	ADTRG0#	P07/IRQ15/ADTRG0#	C4
		P25/CS5#/EDACK1/USB0_DPRPD/MTIOC4C/MTCLKB/PO5/ RxD3/ADTRG0#	M2
EXDMAC0	EDREQ0	P80/EDREQ0/ET_TX_EN/RMII_TXDN/MTIOC3B	R13
		P22/EDREQ0/USB0_DRPD/MTIOC3B/MTCLKC/TMO0/PO2/SCK0	M3
	*15	P55/WAIT#/EDREQ0/ET_EXOUT/MTIOC4D	M6
	EDACK0	P81/EDACK0/ET_ETXD0/RMII_TXD0/MTIOC3D	M11
		P23/EDACK0/USB0_DPUPE/MTIOC3D/MTCLKD/PO3/TxD3	N2
	*15	P54/EDACK0/ET_LINKSTA/MTIOC4B	M7
EXDMAC1	EDREQ1	P82/EDREQ1/ET_ETXD1/RMII_TXD1/MTIOC4A	P11
		P24/CS4#/EDREQ1/USB0_VBUSEN/MTIOC4A/MTCLKA/TMRI1/ PO4/SCK3	P1
	*16	P57/WAIT#/WR3#/BC3#/EDREQ1	N6
	EDACK1	P83/EDACK1/ET_CRS/RMII_CRS_DV/MTIOC4C	R11
		P25/CS5#/EDACK1/USB0_DPRPD/MTIOC4C/MTCLKB/PO5/RxD3/ ADTRG0#	M2
	*16	P56/WR2#/BC2#/EDACK1/MTIOC3C	P7
RSPI0	RSPCKA	PC5/A21/CS2#/WAIT#/ET_ETXD2/MTIC11W/MTCLKD/RSPCKA	N10
	*17	PA5/A5/MTIOC7B/PO21/RSPCKA	J13
	MOSIA	PC6/A22/CS1#/ET_ETXD3/MTIC11V/MTCLKA/MOSIA	M10
	*17	PA6/A6/MTIOC8A/PO22/MOSIA	J15
	MISOA	PC7/A23/CS0#/ET_COL/MTIC11U/MTCLKB/MISOA	R12
	*17	PA7/A7/MTIOC8B/PO23/MISOA	J14
	SSLA0	PC4/A20/CS3#/ET_TX_CLK/MTCLKC/SSLA0	P12
	*17	PA4/A4/MTIOC7A/PO20/SSLA0	H14
	SSLA1	PC0/A16/ET_ERXD3/MTCLKG/SSLA1	M12
	*17	PA0/A0/BC0#/DQM2/MTIOC6A/PO16/SSLA1	F14
	SSLA2	PC1/A17/ET_RXD2/MTCLKH/SSLA2/SCK5	P14
	*17	PA1/A1/DQM3/MTIOC6B/PO17/SSLA2	G15
	SSLA3	PC2/A18/ET_RX_DV/MTCLKE/SSLA3/RxD5	N12
	*17	PA2/A2/MTIOC6C/PO18/SSLA3	H13
RSPI1	RSPCKB	P27/CS7#/USB1_EXICEN/MTIOC2B/PO7/RSPCKB/SCK1	L2
	*18	PE5/D13/RSPCKB/IRQ5	C14
	MOSIB	P26/CS6#/USB1_ID/MTIOC2A/TMO1/PO6/MOSIB/TxD1	N1
	*18	PE6/D14/MOSIB/IRQ6	C15
	MISOB	P30/USB1_DRPD/MTIOC4B/TMRI3/PO8/MISOB/RxD1/IRQ0	L4
	*18	PE7/D15/MISOB/IRQ7	D14
	SSLB0	P31/USB1_DPRPD/MTIOC4D/TMCI2/PO9/SSLB0/IRQ1	K2
	*18	PE4/D12/SSLB0	D13
	SSLB1	P50/WR0#/WR#/SSLB1/TxD2	P10
	*18	PE0/D8/SSLB1	C12

	SSLB2	P51/WR1#/BC1#/WAIT#/SSLB2/SCK2	M8
	*18	PE1/D9/SSLB2	A15
	SSLB3	P52/RD#/SSLB3/RxD2	N8
	*18	PE2/D10/POE9#/SSLB3	B14

*1 to 18 The settings are linked together

Table a-1.2 145-pin TFLGA (the Upper Row of Each Pair is the Default Selection)

Peripheral module	Pin function	Selection of assignment	Pin No.
ICU (External Interrupts)	IRQ2	P32/MTIOC0C/PO10/RTCOUT/CTX0/TxD6/IRQ2	J4
		P12/TMCI1/SCL0/RxD2/IRQ2	L4
	IRQ3	P33/MTIOC0D/PO11/CRX0/RxD6/IRQ3	J1
		P13/TMO3/SDA0/TxD2/IRQ3/ADTRG1#	N4
	IRQ4	P34/MTIOC0A/TMCI3/PO12/SCK6/IRQ4/TRST#	H2
		P14/USB0_OVRCURA/USB0_DPUPE/TMRI2/IRQ4	M5
	IRQ5	PE5/D13/RSPCKB/IRQ5	C11
		P15/MTIOC0B/TMCI2/PO13/SCK3/IRQ5	M4
	IRQ6	PE6/D14/MOSIB/IRQ6	D13
		P16/USB0_VBUS/USB0_OVRCURB/USB0_VBUSEN/MTIOC3C/ TMO2/PO14/RxD3/IRQ6	N2
	IRQ7	PE7/D15/MISOB/IRQ7	D12
		P17/MTIOC3A/PO15/TxD3/IRQ7	L3
	IRQ8	P00/TMRI0/TxD6/IRQ8	E3
		P40/IRQ8/AN0	B4
	IRQ9	P01/TMCI0/RxD6/IRQ9	C1
		P41/IRQ9/AN1	C4
	IRQ10	P02/TMCI1/SCK6/IRQ10	D3
		P42/IRQ10/AN2	A4
	IRQ11	P03/IRQ11/DA0	B1
		P43/IRQ11/AN3	D4
	IRQ13	P05/IRQ13/DA1	C2
		P45/IRQ13/AN5	B5
	IRQ15	P07/IRQ15/ADTRG0#	B2
		P47/IRQ15/AN7	A6
Addres Bus	A16	PC0/A16/ET_ERXD3/MTCLKG/SSLA1	K12
	*1	P90/A16	B6
	A17	PC1/A17/ET_ERXD2/MTCLKH/SSLA2/SCK5	M12
	*1	P91/A17	A7
	A18	PC2/A18/ET_RX_DV/MTCLKE/SSLA3/RxD5	M11
	*1	P92/A18	C6
	A19	PC3/A19/ET_TX_ER/MTCLKF/TxD5	K9
	*1	P93/A19	D7
Bus control	CS0#	P60/CS0#	A11
		PC7/A23/CS0#/ET_COL/MTIC11U/MTCLKB/MISOA	K7
	CS1#	P61/CS1#/SDCS#	C9
		P71/CS1#/ET_MDIO	H11
		PC6/A22/CS1#/ET_ETXD3/MTIC11V/MTCLKA/MOSIA	M9
	CS2#	P62/CS2#/RAS#	A12
		P72/CS2#/ET_MDC	J13
		PC5/A21/CS2#/WAIT#/ET_ETXD2/MTIC11W/MTCLKD/RSPCKA	N10
	CS3#	P63/CS3#/CAS#	C10
		P73/CS3#/ET_WOL	K11

		PC4/A20/CS3#/ET_TX_CLK/MTCLKC/SSLA0	N11
	CS4#	P64/CS4#/WE#	A13
		P74/CS4#/ET_ERXD1/RMII_RXD1	N13
		P24/CS4#/EDREQ1/USB0_VBUSEN/MTIOC4A/MTCLKA/TMRI1/ PO4/SCK3	K2
	CS5#	P65/CS5#/CKE	E10
		P75/CS5#/ET_ERXD0/RMII_RXD0	L11
		P25/CS5#/EDACK1/USB0_DPRPD/MTIOC4C/MTCLKB/PO5/ RxD3/ADTRG0#	L1
	CS6#	P66/CS6#/DQM0	E13
		P76/CS6#/ET_RX_CLK/REF50CK	N12
		P26/CS6#/MTIOC2A/TMO1/PO6/MOSIB/TxD1/TDO	K4
	CS7#	P67/CS7#/DQM1	E11
		P77/CS7#/ET_RX_ER/RMII_RX_ER	L10
		P27/CS7#/MTIOC2B/PO7/RSPCKB/SCK1/TCK	J2
	WAIT#	P55/WAIT#/EDREQ0/ET_EXOUT/MTIOC4D/TRDATA3	M7
		PC5/A21/CS2#/WAIT#/ET_ETXD2/MTIC11W/MTCLKD/RSPCKA	N10
		P51/WR1#/BC1#/WAIT#/SSLB2/SCK2	N8
MTU0-5	MTCLKA	P24/CS4#/EDREQ1/USB0_VBUSEN/MTIOC4A/MTCLKA/TMRI1/ PO4/SCK3	K2
	*2	PC6/A22/CS1#/ET_ETXD3/MTIC11V/MTCLKA/MOSIA	M9
	MTCLKB	P25/CS5#/EDACK1/USB0_DPRPD/MTIOC4C/MTCLKB/PO5/ RxD3/ADTRG0#	L1
	*2	PC7/A23/CS0#/ET_COL/MTIC11U/MTCLKB/MISOA	K7
	MTCLKC	P22/EDREQ0/USB0_DRPD/MTIOC3B/MTCLKC/TMO0/PO2/SCK0	L2
	*2	PC4/A20/CS3#/ET_TX_CLK/MTCLKC/SSLA0	N11
	MTCLKD	P23/EDACK0/USB0_DPUPE/MTIOC3D/MTCLKD/PO3/TxD3	M1
	*2	PC5/A21/CS2#/WAIT#/ET_ETXD2/MTIC11W/MTCLKD/RSPCKA	N10
MTU6-11	MTCLKE	PC2/A18/ET_RX_DV/MTCLKE/SSLA3/RxD5	M11
	*3	PB4/A12/MTIOC10A/MTCLKE/PO28	J11
	MTCLKF	PC3/A19/ET_TX_ER/MTCLKF/TxD5	K9
	*3	PB5/A13/MTIOC10C/MTCLKF/PO29	J12
	MTCLKG	PC0/A16/ET_ERXD3/MTCLKG/SSLA1	K12
	*3	PB2/A10/MTIOC9B/MTCLKG/PO26	J10
	MTCLKH	PC1/A17/ET_ERXD2/MTCLKH/SSLA2/SCK5	M12
	*3	PB3/A11/MTIOC9D/MTCLKH/PO27	K13
MTU3	MTIOC3B	P22/EDREQ0/USB0_DRPD/MTIOC3B/MTCLKC/TMO0/PO2/SCK0	L2
	*4	P80/EDREQ0/ET_TX_EN/RMII_TXD_EN/MTIOC3B/TRDATA0	M10
	MTIOC3C	P16/USB0_VBUS/USB0_OVRCURB/USB0_VBUSEN/MTIOC3C/ TMO2/PO14/RxD3/IRQ6	N2
		P56/EDACK1/MTIOC3C	L6
	MTIOC3D	P23/EDACK0/USB0_DPUPE/MTIOC3D/MTCLKD/PO3/TxD3	M1
	*4	P81/EDACK0/ET_ETXD0/RMII_TXD0/MTIOC3D/TRDATA1	L9
MTU4	MTIOC4A	P24/CS4#/EDREQ1/USB0_VBUSEN/MTIOC4A/MTCLKA/TMRI1/ PO4/SCK3	K2
	*5	P82/EDREQ1/ET_ETXD1/RMII_TXD1/MTIOC4A/TRSYNC	K8
	MTIOC4B	P30/MTIOC4B/TMRI3/PO8/RxD1/MISOB/IRQ0/TDI	K1
	*6	P54/EDACK0/ET_LINKSTA/MTIOC4B/TRDATA2	N7
	MTIOC4C	P25/CS5#/EDACK1/USB0_DPRPD/MTIOC4C/MTCLKB/PO5/ RxD3/ADTRG0#	L1
	*5	P83/EDACK1/ET_CRS/RMII_CRS_DV/MTIOC4C/TRCLK	L8
	MTIOC4D	P31/MTIOC4D/TMCI2/PO9/SSLB0/IRQ1/TMS	K3
	*6	P55/WAIT#/EDREQ0/ET_EXOUT/MTIOC4D/TRDATA3	M7

MTU11	MTIC11U	PC7/A23/CS0#/ET_COL/MTIC11U/MTCLKB/MISOA	K7
	*7	PD4/D4/MTIC11U/POE3#	D8
	MTIC11V	PC6/A22/CS1#/ET_ETXD3/MTIC11V/MTCLKA/MOSIA	M9
	*7	PD3/D3/MTIC11V/POE4#	A9
	MTIC11W	PC5/A21/CS2#/WAIT#/ET_ETXD2/MTIC11W/MTCLKD/RSPCKA	N10
	*7	PD2/D2/MTIC11W/POE5#	C7
TMR0	TMCI0	P01/TMCI0/RxD6/IRQ9	C1
	*8	P21/USB0_EXICEN/MTIOC1B/TMCI0/PO1/SCL1/RxD0	N1
	TMRI0	P00/TMRI0/TxD6/IRQ8	E3
	*8	P20/USB0_ID/MTIOC1A/TMRI0/PO0/SDA1/TxD0	M2
TMR1	TMCI1	P02/TMCI1/SCK6/IRQ10	D3
		P12/TMCI1/SCL0/RxD2/IRQ2	L4
TMR2	TMCI2	P15/MTIOC0B/TMCI2/PO13/SCK3/IRQ5	M4
		P31/MTIOC4D/TMCI2/PO9/SSLB0/IRQ1/TMS	K3
SCI2	TxD2	P13/TMO3/SDA0/TxD2/IRQ3/ADTRG1#	N4
	*9	P50/WR0#/WR#/SSLB1/TxD2	M8
	RxD2	P12/TMCI1/SCL0/RxD2/IRQ2	L4
	*9	P52/RD#/SSLB3/RxD2	L7
SCI3	TxD3	P17/MTIOC3A/PO15/TxD3/IRQ7	L3
	*10	P23/EDACK0/USB0_DPUPE/MTIOC3D/MTCLKD/PO3/TxD3	M1
	RxD3	P16/USB0_VBUS/USB0_OVRCURB/USB0_VBUSEN/MTIOC3C/ TMO2/PO14/RxD3/IRQ6	N2
	*10	P25/CS5#/EDACK1/USB0_DPRPD/MTIOC4C/MTCLKB/PO5/ RxD3/ADTRG0#	L1
	SCK3	P15/MTIOC0B/TMCI2/PO13/SCK3/IRQ5	M4
	*10	P24/CS4#/EDREQ1/USB0_VBUSEN/MTIOC4A/MTCLKA/TMRI1/ PO4/SCK3	K2
SCI6	TxD6	P00/TMRI0/TxD6/IRQ8	E3
	*11	P32/MTIOC0C/PO10/RTCOUT/CTX0/TxD6/IRQ2	J4
	RxD6	P01/TMCI0/RxD6/IRQ9	C1
	*11	P33/MTIOC0D/PO11/CRX0/RxD6/IRQ3	J1
	SCK6	P02/TMCI1/SCK6/IRQ10	D3
	*11	P34/MTIOC0A/TMCI3/PO12/SCK6/IRQ4/TRST#	H2
AD0, S12AD0	ADTRG0#	P07/IRQ15/ADTRG0#	B2
		P25/CS5#/EDACK1/USB0_DPRPD/MTIOC4C/MTCLKB/PO5/ RxD3/ADTRG0#	L1
EXDMAC0	EDREQ0	P80/EDREQ0/ET_TX_EN/RMII_TXD_EN/MTIOC3B/TRDATA0	M10
		P22/EDREQ0/USB0_DRPD/MTIOC3B/MTCLKC/TMO0/PO2/SCK0	L2
	*12	P55/WAIT#/EDREQ0/ET_EXOUT/MTIOC4D/TRDATA3	M7
	EDACK0	P81/EDACK0/ET_ETXD0/RMII_TXD0/MTIOC3D/TRDATA1	L9
		P23/EDACK0/USB0_DPUPE/MTIOC3D/MTCLKD/PO3/TxD3	M1
	*12	P54/EDACK0/ET_LINKSTA/MTIOC4B/TRDATA2	N7
EXDMAC1	EDREQ1	P82/EDREQ1/ET_ETXD1/RMII_TXD1/MTIOC4A/TRSYNC	K8
		P24/CS4#/EDREQ1/USB0_VBUSEN/MTIOC4A/MTCLKA/TMRI1/ PO4/SCK3	K2
	*13	P24/CS4#/EDREQ1/USB0_VBUSEN/MTIOC4A/MTCLKA/TMRI1/ PO4/SCK3	K2
	EDACK1	P83/EDACK1/ET_CRS/RMII_CRS_DV/MTIOC4C/TRCLK	L8
		P25/CS5#/EDACK1/USB0_DPRPD/MTIOC4C/MTCLKB/PO5/ RxD3/ADTRG0#	L1
	*13	P56/EDACK1/MTIOC3C	L6
RSPI0	RSPCKA	PC5/A21/CS2#/WAIT#/ET_ETXD2/MTIC11W/MTCLKD/RSPCKA	N10
	*14	PA5/A5/MTIOC7B/PO21/RSPCKA	G11

	MOSIA	PC6/A22/CS1#/ET_ETXD3/MTIC11V/MTCLKA/MOSIA	M9
	*14	PA6/A6/MTIOC8A/PO22/MOSIA	G12
	MISOA	PC7/A23/CS0#/ET_COL/MTIC11U/MTCLKB/MISOA	K7
	*14	PA7/A7/MTIOC8B/PO23/MISOA	H13
	SSLA0	PC4/A20/CS3#/ET_TX_CLK/MTCLKC/SSLA0	N11
	*14	PA4/A4/MTIOC7A/PO20/SSLA0	G13
	SSLA1	PC0/A16/ET_ERXD3/MTCLKG/SSLA1	K12
	*14	PA0/A0/BC0#/MTIOC6A/PO16/SSLA1	E12
	SSLA2	PC1/A17/ET_ERXD2/MTCLKH/SSLA2/SCK5	M12
	*14	PA1/A1/MTIOC6B/PO17/SSLA2	F10
	SSLA3	PC2/A18/ET_RX_DV/MTCLKE/SSLA3/RxD5	M11
	*14	PA2/A2/MTIOC6C/PO18/SSLA3	F13
RSPI1	RSPCKB	P27/CS7#/MTIOC2B/PO7/RSPCKB/SCK1/TCK	J2
	*15	PE5/D13/RSPCKB/IRQ5	C11
	MOSIB	P26/CS6#/MTIOC2A/TMO1/PO6/MOSIB/TxD1/TDO	K4
	*15	PE6/D14/MOSIB/IRQ6	D13
	MISOB	P30/MTIOC4B/TMRI3/PO8/RxD1/MISOB/IRQ0/TDI	K1
	*15	PE7/D15/MISOB/IRQ7	D12
	SSLB0	P31/MTIOC4D/TMCI2/PO9/SSLB0/IRQ1/TMS	K3
	*15	PE4/D12/SSLB0	B13
	SSLB1	P50/WR0#/WR#/SSLB1/TxD2	M8
	*15	PE0/D8/SSLB1	B10
	SSLB2	P51/WR1#/BC1#/WAIT#/SSLB2/SCK2	N8
	*15	PE1/D9/SSLB2	B12
	SSLB3	P52/RD#/SSLB3/RxD2	L7
	*15	PE2/D10/POE9#/SSLB3	B11

*1 to 15 The settings are linked together

Table a-1.2 144-pin LQFP (the Upper Row of Each Pair is the Default Selection)

Peripheral module	Pin function	Selection of assignment	Pin No.
ICU (External Interrupts)	IRQ2	P32/PO10/MTIOC0C/TxD6/CTX0/IRQ2/RTCOUT	27
		P12/TMCI1/RxD2/SCL0/IRQ2	45
	IRQ3	P33/PO11/MTIOC0D/RxD6/CRX0/IRQ3	26
		P13/ADTRG1#/TMO3/TxD2/SDA0/IRQ3	44
	IRQ4	P34/PO12/MTIOC0A/TMCI3/SCK6/IRQ4/TRST#	25
		P14/TMRI2/IRQ4/USB0_OVRCURA/USB0_DPUPE	43
	IRQ5	PE5/D13/RSPCKB/IRQ5	106
		P15/PO13/MTIOC0B/TMCI2/SCK3/IRQ5	42
	IRQ6	PE6/D14/MOSIB/IRQ6	102
		P16/PO14/MTIOC3C/TMO2/RxD3/IRQ6/USB0_VBUS/ USB0_OVRCURB/USB0_VBUSEN	40
	IRQ7	PE7/D15/MISOB/IRQ7	101
		P17/PO15/MTIOC3A/TxD3/IRQ7	38
	IRQ8	P00/TMRI0/TxD6/IRQ8	8
		P40/AN0/IRQ8	141
	IRQ9	P01/TMCI0/RxD6/IRQ9	7
		P41/AN1/IRQ9	139
	IRQ10	P02/TMCI1/SCK6/IRQ10	6
		P42/AN2/IRQ10	138
	IRQ11	P03/DA0/IRQ11	4

		P43/AN3/IRQ11	137
	IRQ13	P05/DA1/IRQ13	2
		P45/AN5/IRQ13	135
	IRQ15	P07/ADTRG0#/IRQ15	144
		P47/AN7/IRQ15	133
Addres Bus	A16 *1	PC0/A16/MTCLKG/SSLA1/ET_ERXD3	75
		P90/A16	131
	A17 *1	PC1/A17/MTCLKH/SCK5/SSLA2/ET_ERXD2	73
		P91/A17	129
	A18 *1	PC2/A18/MTCLKE/RxD5/SSLA3/ET_RX_DV	70
		P92/A18	128
	A19 *1	PC3/A19/MTCLKF/TxD5/ET_TX_ER	67
		P92/A19	127
Bus control	CS0#	P60/CS0#	117
		PC7/A23/CS0#/MTIC11U/MTCLKB/MISOA/ET_COL	60
	CS1#	P61/CS1#/SDCS#	115
		P71/CS1#/ET_MDIO	86
		PC6/A22/CS1#/MTIC11V/MTCLKA/MOSIA/ET_ETXD3	61
	CS2#	P62/CS2#/RAS#	114
		P72/CS2#/ET_MDC	85
		PC5/A21/CS2#/WAIT#/MTIC11W/MTCLKD/RSPCKA/ET_ETXD2	62
	CS3#	P63/CS3#/CAS#	113
		P73/CS3#/ET_WOL	77
		PC4/A20/CS3#/MTCLKC/SSLA0/ET_TX_CLK	66
	CS4#	P64/CS4#/WE#	112
		P74/CS4#/ET_ERXD1/RMII_RXD1	72
		P24/CS4#/EDREQ1/PO4/MTIOC4A/MTCLKA/TMRI1/ SCK3/USB0_VBUSEN	33
	CS5#	P65/CS5#/CKE	100
		P75/CS5#/ET_ERXD0/RMII_RXD0	71
		P25/CS5#/EDACK1/ADTRG0#/PO5/MTIOC4C/MTCLKB/ RxD3/USB0_DPRPD	32
	CS6#	P66/CS6#/DQM0	99
		P76/CS6#/ET_RX_CLK/REF50CK	69
		P26/CS6#/PO6/MTIOC2A/TMO1/TxD1/MOSIB/TDO	31
	CS7#	P67/CS7#/DQM1	98
		P77/CS7#/ET_RX_ER/RMII_RX_ER	68
		P27/CS7#/PO7/MTIOC2B/SCK1/RSPCKB/TCK	30
	WAIT#	P55/WAIT#/EDREQ0/MTIOC4D/ET_EXOUT/TRDATA3	51
		PC5/A21/CS2#/WAIT#/MTIC11W/MTCLKD/RSPCKA/ET_ETXD2	62
		P51/WR1#/BC1#/WAIT#/SCK2/SSLB2	55
MTU0-5	MTCLKA *2	P24/CS4#/EDREQ1/PO4/MTIOC4A/MTCLKA/TMRI1/ SCK3/USB0_VBUSEN	33
		PC6/A22/CS1#/MTIC11V/MTCLKA/MOSIA/ET_ETXD3	61
	MTCLKB *2	P25/CS5#/EDACK1/ADTRG0#/PO5/MTIOC4C/MTCLKB/ RxD3/USB0_DPRPD	32
		PC7/A23/CS0#/MTIC11U/MTCLKB/MISOA/ET_COL	60
	MTCLKC *2	P22/EDREQ0/PO2/MTIOC3B/MTCLKC/TMO0/SCK0/ USB0_DRPD	35
		PC4/A20/CS3#/MTCLKC/SSLA0/ET_TX_CLK	66
	MTCLKD *2	P23/EDACK0/PO3/MTIOC3D/MTCLKD/TxD3/USB0_DPUPE	34
		PC5/A21/CS2#/WAIT#/MTIC11W/MTCLKD/RSPCKA/ ET_ETXD2	62

MTU6-11	MTCLKE	PC2/A18/MTCLKE/RxD5/SSLA3/ET_RX_DV	70
	*3	PB4/A12/PO28/MTIOC10A/MTCLKE	81
	MTCLKF	PC3/A19/MTCLKF/TxD5/ET_TX_ER	67
	*3	PB5/A13/PO29/MTIOC10C/MTCLKF	80
	MTCLKG	PC0/A16/MTCLKG/SSLA1/ET_ERXD3	75
	*3	PB2/A10/PO26/MTIOC9B/MTCLKG	83
	MTCLKH	PC1/A17/MTCLKH/SCK5/SSLA2/ET_ERXD2	73
	*3	PB3/A11/PO27/MTIOC9D/MTCLKH	82
MTU3	MTIOC3B	P22/EDREQ0/PO2/MTIOC3B/MTCLKC/TMO0/SCK0/ USB0_DRPD	35
	*4	P80/EDREQ0/MTIOC3B/ET_TX_EN/RMII_TXD_EN/ TRDATA0	65
	MTIOC3C	P16/PO14/MTIOC3C/TMO2/RxD3/IRQ6/USB0_VBUS/ USB0_OVRCURB/USB0_VBUSEN	40
		P56/EDACK1/MTIOC3C	50
	MTIOC3D	P23/EDACK0/PO3/MTIOC3D/MTCLKD/TxD3/USB0_DPUPE	34
	*4	P81/EDACK0/MTIOC3D/ET_ETXD0/RMII_TXD0/TRDATA1	64
MTU4	MTIOC4A	P24/CS4#/EDREQ1/PO4/MTIOC4A/MTCLKA/TMRI1/ SCK3/USB0_VBUSEN	33
	*5	P82/EDREQ1/MTIOC4A/ET_ETXD1/RMII_TXD1/TRSYNC	63
	MTIOC4B	P30/PO8/MTIOC4B/TMRI3/RxD1/MISOB/IRQ0/TDI	29
	*6	P54/EDACK0/MTIOC4B/ET_LINKSTA/TRDATA2	52
	MTIOC4C	P25/CS5#/EDACK1/ADTRG0#/PO5/MTIOC4C/MTCLKB/ RxD3/USB0_DPRPD	32
	*5	P83/EDACK1/MTIOC4C/ET_CRS/RMII_CRS_DV/TRCLK	58
	MTIOC4D	P31/PO9/MTIOC4D/TMCI2/SSLB0/IRQ1/TMS	28
	*6	P55/WAIT#/EDREQ0/MTIOC4D/ET_EXOUT/TRDATA3	51
MTU11	MTIC11U	PC7/A23/CS0#/MTIC11U/MTCLKB/MISOA/ET_COL	60
	*7	PD4/D4/MTIC11U/POE3#	122
	MTIC11V	PC6/A22/CS1#/MTIC11V/MTCLKA/MOSIA/ET_ETXD3	61
	*7	PD3/D3/MTIC11V/POE4#	123
	MTIC11W	PC5/A21/CS2#/WAIT#/MTIC11W/MTCLKD/RSPCKA/ ET_ETXD2	62
	*7	PD2/D2/MTIC11W/POE5#	124
TMR0	TMCI0	P01/TMCI0/RxD6/IRQ9	7
	*8	P21/PO1/MTIOC1B/TMCI0/RxD0/SCL1/USB0_EXICEN	36
	TMRI0	P00/TMRI0/TxD6/IRQ8	8
	*8	P20/PO0/MTIOC1A/TMRI0/TxD0/SDA1/USB0_ID	37
TMR1	TMCI1	P02/TMCI1/SCK6/IRQ10	6
		P12/TMCI1/RxD2/SCL0/IRQ2	45
TMR2	TMCI2	P15/PO13/MTIOC0B/TMCI2/SCK3/IRQ5	42
		P31/PO9/MTIOC4D/TMCI2/SSLB0/IRQ1/TMS	28
SCI2	TxD2	P13/ADTRG1#/TMO3/TxD2/SDA0/IRQ3	44
	*9	P50/WR0#/WR#/TxD2/SSLB1	56
	RxD2	P12/TMCI1/RxD2/SCL0/IRQ2	45
	*9	P52/RD#/RxD2/SSLB3	54
SCI3	TxD3	P17/PO15/MTIOC3A/TxD3/IRQ7	38
	*10	P23/EDACK0/PO3/MTIOC3D/MTCLKD/TxD3/USB0_DPUPE	34
	RxD3	P16/PO14/MTIOC3C/TMO2/RxD3/IRQ6/USB0_VBUS /USB0_OVRCURB/USB0_VBUSEN	40
	*10	P25/CS5#/EDACK1/ADTRG0#/PO5/MTIOC4C/MTCLKB/ RxD3/USB0_DPRPD	32
	SCK3	P15/PO13/MTIOC0B/TMCI2/SCK3/IRQ5	42

	*10	P24/CS4#/EDREQ1/PO4/MTIOC4A/MTCLKA/TMRI1/ SCK3/USB0_VBUSEN	33
SCI6	TxD6	P00/TMRI0/TxD6/IRQ8	8
	*11	P32/PO10/MTIOC0C/TxD6/CTX0/IRQ2/RTCOUT	27
	RxD6	P01/TMCI0/RxD6/IRQ9	7
	*11	P33/PO11/MTIOC0D/RxD6/CRX0/IRQ3	26
	SCK6	P02/TMCI1/SCK6/IRQ10	6
	*11	P34/PO12/MTIOC0A/TMCI3/SCK6/IRQ4/TRST#	25
AD0, S12AD0	ADTRG0#	P07/ADTRG0#/IRQ15	144
		P25/CS5#/EDACK1/ADTRG0#/PO5/MTIOC4C/MTCLKB/ RxD3/USB0_DPRPD	32
EXDMAC0	EDREQ0	P80/EDREQ0/MTIOC3B/ET_TX_EN/RMII_TXD_EN/TRDATA0	65
		P22/EDREQ0/PO2/MTIOC3B/MTCLKC/TMO0/SCK0/USB0_DRPD	35
	*12	P55/WAIT#/EDREQ0/MTIOC4D/ET_EXOUT/TRDATA3	51
	EDACK0	P81/EDACK0/MTIOC3D/ET_ETXD0/RMII_TXD0/TRDATA1	64
		P23/EDACK0/PO3/MTIOC3D/MTCLKD/TxD3/ USB0_DPUPE	34
	*12	P54/EDACK0/MTIOC4B/ET_LINKSTA/TRDATA2	52
EXDMAC1	EDREQ1	P82/EDREQ1/MTIOC4A/ET_ETXD1/RMII_TXD1/TRSYNC	63
		P24/CS4#/EDREQ1/PO4/MTIOC4A/MTCLKA/TMRI1/ SCK3/USB0_VBUSEN	33
	*13	P24/CS4#/EDREQ1/PO4/MTIOC4A/MTCLKA/TMRI1/ SCK3/USB0_VBUSEN	33
	EDACK1	P83/EDACK1/MTIOC4C/ET_CRS/RMII_CRS_DV/TRCLK	58
		P25/CS5#/EDACK1/ADTRG0#/PO5/MTIOC4C/MTCLKB/ RxD3/USB0_DPRPD	32
	*13	P56/EDACK1/MTIOC3C	50
RSPI0	RSPCKA	PC5/A21/CS2#/WAIT#/MTIC11W/MTCLKD/RSPCKA/ET_ETXD2	62
	*14	PA5/A5/PO21/MTIOC7B/RSPCKA	90
	MOSIA	PC6/A22/CS1#/MTIC11V/MTCLKA/MOSIA/ET_ETXD3	61
	*14	PA6/A6/PO22/MTIOC8A/MOSIA	89
	MISOA	PC7/A23/CS0#/MTIC11U/MTCLKB/MISOA/ET_COL	60
	*14	PA7/A7/PO23/MTIOC8B/MISOA	88
	SSLA0	PC4/A20/CS3#/MTCLKC/SSLA0/ET_TX_CLK	66
	*14	PA4/A4/PO20/MTIOC7A/SSLA0	92
	SSLA1	PC0/A16/MTCLKG/SSLA1/ET_ERXD3	75
	*14	PA0/A0/BC0#/PO16/MTIOC6A/SSLA1	97
	SSLA2	PC1/A17/MTCLKH/SCK5/SSLA2/ET_ERXD2	73
	*14	PA1/A1/PO17/MTIOC6B/SSLA2	96
	SSLA3	PC2/A18/MTCLKE/RxD5/SSLA3/ET_RX_DV	70
	*14	PA2/A2/PO18/MTIOC6C/SSLA3	95
RSPI1	RSPCKB	P27/CS7#/PO7/MTIOC2B/SCK1/RSPCKB/TCK	30
	*15	PE5/D13/RSPCKB/IRQ5	106
	MOSIB	P26/CS6#/PO6/MTIOC2A/TMO1/TxD1/MOSIB/TDO	31
	*15	PE6/D14/MOSIB/IRQ6	102
	MISOB	P30/PO8/MTIOC4B/TMRI3/RxD1/MISOB/IRQ0/TDI	29
	*15	PE7/D15/MISOB/IRQ7	101
	SSLB0	P31/PO9/MTIOC4D/TMCI2/SSLB0/IRQ1/TMS	28
	*15	PE4/D12/SSLB0	107
	SSLB1	P50/WR0#/WR#/TxD2/SSLB1	56
	*15	PE0/D8/SSLB1	111
	SSLB2	P51/WR1#/BC1#/WAIT#/SCK2/SSLB2	55
	*15	PE1/D9/SSLB2	110

	SSLB3	P52/RD#/RxD2/SSLB3	54
	*15	PE2/D10/SSLB3/POE9#	109

*1 to 15 The settings are linked together

Table a-1.2 100-pin LQFP (the Upper Row of Each Pair is the Default Selection)

Peripheral module	Pin function	Selection of assignment	Pin No.
ICU (External Interrupts)	IRQ2	P32/PO10/MTIOC0C/TxD6/CTX0/IRQ2/RTCOUT	18
		P12/TMCI1/RxD2/SCL0/IRQ2	34
	IRQ3	P33/PO11/MTIOC0D/RxD6/CRX0/IRQ3	17
		P13/ADTRG1#/PO13/MTIOC0B/TMO3/TxD2/SDA0/IRQ3	33
	IRQ4	P34/PO12/MTIOC0A/TMCI3/SCK6/IRQ4/TRST#	16
		P14/PO15/MTIOC3A/TMRI2/IRQ4/USB0_OVRCURA/ USB0_DPUPE	32
	IRQ6	PE6/D14/MOSIB/IRQ6	72
		P16/PO14/MTIOC3C/TMO2/IRQ6/USB0_VBUS/ USB0_OVRCURB/USB0_VBUSEN	30
	IRQ13	P05/DA1/IRQ13	100
		P45/AN5/IRQ13	89
	IRQ15	P07/ADTRG0#/IRQ15	98
		P47/AN7/IRQ15	87
Bus control	WAIT#	P55/WAIT#/MTIOC4D	39
		PC5/A21/CS2#/WAIT#/MTIC11W/MTCLKD/RSPCKA/ET_ETXD2	47
		P51/WR1#/BC1#/WAIT#/SCK2/SSLB2	43
MTU0-5	MTCLKA	P24/CS4#/PO4/MTIOC4A/MTCLKA/TMRI1/SCK3/ USB0_VBUSEN	24
	*1	PC6/A22/CS1#/MTIC11V/MTCLKA/MOSIA/ET_ETXD3	46
	MTCLKB	P25/CS5#/ADTRG0#/PO5/MTIOC4C/MTCLKB/RxD3/ USB0_DPRPD	23
	*1	PC7/A23/CS0#/MTIC11U/MTCLKB/MISOA/ET_COL	45
	MTCLKC	P22/PO2/MTIOC3B/MTCLKC/TMO0/SCK0/USB0_DRPD	26
	*1	PC4/A20/CS3#/MTCLKC/SSLA0/ET_TX_CLK	48
	MTCLKD	P23/PO3/MTIOC3D/MTCLKD/TxD3/USB0_DPUPE	25
	*1	PC5/A21/CS2#/WAIT#/MTIC11W/MTCLKD/RSPCKA/ET_ETXD2	47
MTU6-11	MTCLKE	PC2/A18/MTCLKE/RxD5/SSLA3/ET_RX_DV	50
	*2	PB4/A12/PO28/MTIOC10A/MTCLKE/ET_TX_EN/RMII_TXD_EN	56
	MTCLKF	PC3/A19/MTCLKF/TxD5/ET_TX_ER	49
	*2	PB5/A13/PO29/MTIOC10C/MTCLKF/ET_ETXD0/RMII_TXD0	55
	MTCLKG	PC0/A16/MTCLKG/SSLA1/ET_ERXD3	52
	*2	PB2/A10/PO26/MTIOC9B/MTCLKG/ET_RX_CLK/REF50CK	58
	MTCLKH	PC1/A17/MTCLKH/SCK5/SSLA2/ET_ERXD2	51
	*2	PB3/A11/PO27/MTIOC9D/MTCLKH/ET_RX_ER/RMII_RX_ER	57
MTU4	MTIOC4B	P30/PO8/MTIOC4B/TMRI3/RxD1/MISOB/IRQ0/TDI	20
	*3	P54/MTIOC4B	40
	MTIOC4D	P31/PO9/MTIOC4D/TMCI2/SSLB0/IRQ1/TMS	19
	*3	P55/WAIT#/MTIOC4D	39
MTU11	MTIC11U	PC7/A23/CS0#/MTIC11U/MTCLKB/MISOA/ET_COL	45
	*4	PD4/D4/MTIC11U/POE3#	82
	MTIC11V	PC6/A22/CS1#/MTIC11V/MTCLKA/MOSIA/ET_ETXD3	46
	*4	PD3/D3/MTIC11V/POE4#	83
	MTIC11W	PC5/A21/CS2#/WAIT#/MTIC11W/MTCLKD/RSPCKA/	47

		ET_ETXD2	
	*4	PD2/D2/MTIC11W/POE5#	84
SCI2	TxD2	P13/ADTRG1#/PO13/MTIOC0B/TMO3/TxD2/SDA0/IRQ3	33
	*5	P50/WR0#/WR#/TxD2/SSLB1	44
	RxD2	P12/TMCI1/RxD2/SCL0/IRQ2	34
	*5	P52/RD#/RxD2/SSLB3	42
AD0, S12AD0	ADTRG0#	P07/ADTRG0#/IRQ15	98
		P25/CS5#/ADTRG0#/PO5/MTIOC4C/MTCLKB/RxD3/ USB0_DPRPD	23
RSPI0	RSPCKA	PC5/A21/CS2#/WAIT#/MTIC11W/MTCLKD/RSPCKA/ET_ETXD2	47
	*6	PA5/A5/PO21/MTIOC7B/RSPCKA/ET_LINKSTA	65
	MOSIA	PC6/A22/CS1#/MTIC11V/MTCLKA/MOSIA/ET_ETXD3	46
	*6	PA6/A6/PO22/MTIOC8A/MOSIA/ET_EXOUT	64
	MISOA	PC7/A23/CS0#/MTIC11U/MTCLKB/MISOA/ET_COL	45
	*6	PA7/A7/PO23/MTIOC8B/MISOA/ET_WOL	63
	SSLA0	PC4/A20/CS3#/MTCLKC/SSLA0/ET_TX_CLK	48
	*6	PA4/A4/PO20/MTIOC7A/SSLA0/ET_MDC	66
	SSLA1	PC0/A16/MTCLKG/SSLA1/ET_ERXD3	52
	*6	PA0/A0/BC0#/PO16/MTIOC6A/SSLA1	70
	SSLA2	PC1/A17/MTCLKH/SCK5/SSLA2/ET_ERXD2	51
	*6	PA1/A1/PO17/MTIOC6B/SSLA2	69
	SSLA3	PC2/A18/MTCLKE/RxD5/SSLA3/ET_RX_DV	50
	*6	PA2/A2/PO18/MTIOC6C/SSLA3	68
RSPI1	RSPCKB	P27/CS7#/PO7/MTIOC2B/SCK1/RSPCKB/TCK	21
	*7	PE5/D13/RSPCKB/IRQ5	73
	MOSIB	P26/CS6#/PO6/MTIOC2A/TMO1/TxD1/MOSIB/TDO	22
	*7	PE6/D14/MOSIB/IRQ6	72
	MISOB	P30/PO8/MTIOC4B/TMRI3/RxD1/MISOB/IRQ0/TDI	20
	*7	PE7/D15/MISOB/IRQ7	71
	SSLB0	P31/PO9/MTIOC4D/TMCI2/SSLB0/IRQ1/TMS	19
	*7	PE4/D12/SSLB0	74
	SSLB1	P50/WR0#/WR#/TxD2/SSLB1	44
	*7	PE0/D8/SSLB1	78
	SSLB2	P51/WR1#/BC1#/WAIT#/SCK2/SSLB2	43
	*7	PE1/D9/SSLB2	77
	SSLB3	P52/RD#/RxD2/SSLB3	42
	*7	PE2/D10/SSLB3/POE9#	76

*1 to 7 The settings are linked together

RX62N Group
Peripheral Driver Generator
Reference Manual

Publication Date: May 16, 2014 Rev.1.04

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

RX62N Group
Peripheral Driver Generator
Reference Manual

Renesas Electronics Corporation

R20UT0565EJ0104