User's Manual

RX610 Group

Peripheral Driver Generator
Reference Manual

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corporation without notice. Please review the latest information published by Renesas Electronics Corporation through various means, including the Renesas Electronics Corporation website (http://www.renesas.com).

Renesas Electronics
www.renesas.com

Rev.1.01 Feb 2011

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. 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.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. 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.
5. When exporting the 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. You should not use Renesas Electronics products or the 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. 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.
6. 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.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. 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 categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. 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 an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "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.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. 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.
9. 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 system manufactured by you.
10. 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.
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 RX610. 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 RX610 group, peripheral modules and endian.

(1) Products

Part No.	Package
R5F56108VNFPP	LQP0144KAA
R5F56107VNFPP	LQP0144KAA
R5F56106VNFPP	LQP0144KAA
R5F56104VNFPP	LQP0144KAA

(2) Products

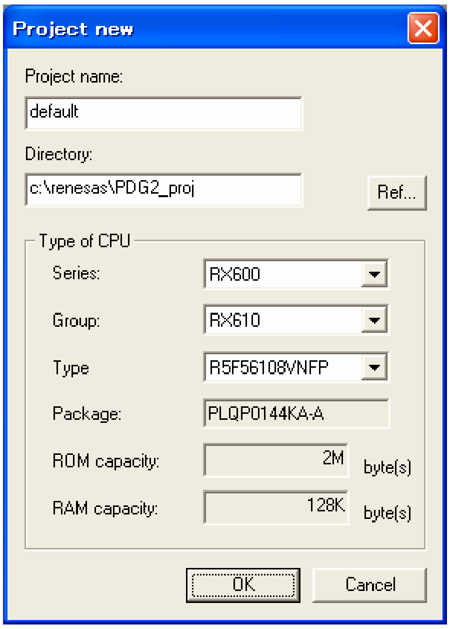
- Clock Generation Circuit
- Interrupt Control Unit (ICU), Exceptions
- DMA Controller (DMAC)
- I/O Ports
- 16-Bit Timer Pulse Unit (TPU)
- 8-Bit Timer (TMR)
- Compare Match Timer (CMT)
- Serial Communications Interface (SCI)
- I2C Bus Interface (RIIC)
- A/D Converter

(3) Endian

Little endian

2. Creating a new project

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



Project name: default

Directory: c:\renesas\PDG2_proj

Type of CPU

Series: RX600

Group: RX610

Type R5F56108VNFP

Package: PLQP0144KA-A

ROM capacity: 2M byte(s)

RAM capacity: 128K byte(s)

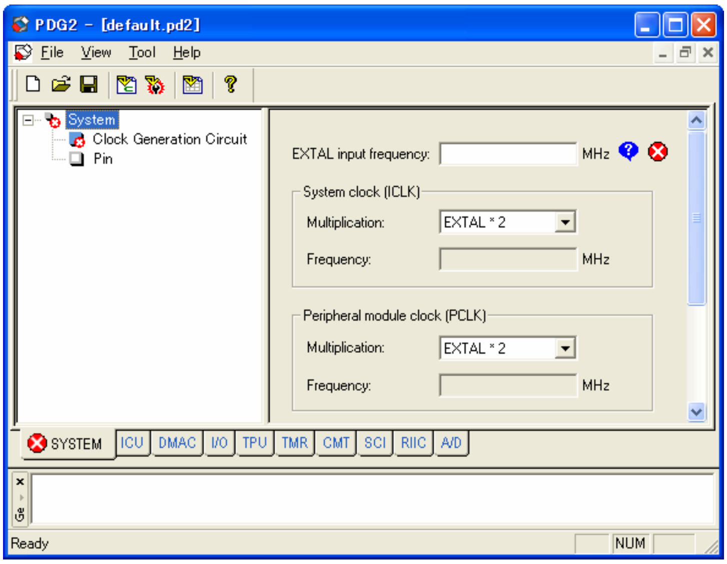
Fig 2.1 New project dialog box

For RX610 group, select [RX600] as a series and select [RX610] 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.



EXTAL input frequency: MHz

System clock (ICLK)

Multiplication: EXTAL * 2

Frequency: MHz

Peripheral module clock (PCLK)

Multiplication: EXTAL * 2

Frequency: MHz

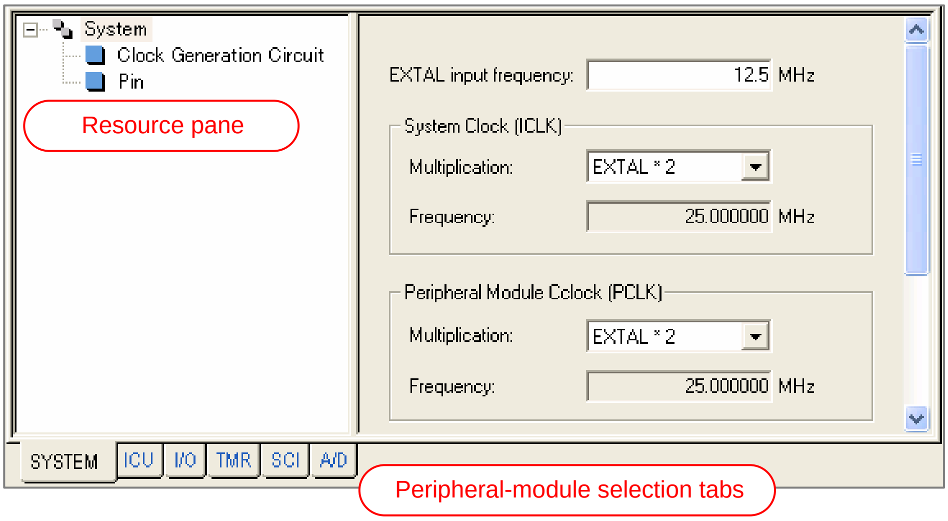
Fig 2.2 Error display of new project

Set the frequency of the lock to be used here.

3. Setting the Peripheral Modules

3.1 Peripheral Module Setting Windows

Figure 3.1 shows the example of peripheral module setting window display.



EXTAL input frequency: 12.5 MHz

System Clock (ICLK)

Multiplication: EXTAL * 2

Frequency: 25.000000 MHz

Peripheral Module Cclock (PCLK)

Multiplication: EXTAL * 2

Frequency: 25.000000 MHz

Figure 3.1 The example of peripheral module setting window display

The correspondences of the resources to a peripheral modules or functions are shown in table 3.1.

Table 3.1 The correspondences of the resources to a peripheral modules or functions

Peripheral-module selection tab	Resource pane	Corresponding Peripheral Module or Function
SYSTEM	Clock Generation Circuit	Clock Generation Circuit
	Pin	Pinfunctions
ICU	Interrupts	Interrupt Control Unit (ICU) (Fastinterrupt, NMI, IRQ0 to IRQ15)
	Exceptions	Exceptions
DMAC	DMAC0 to DMAC3	DMA Controller (DMAC) Channel 0 to Channel 3
I/O	Port0 to PortE	I/O Port 0 to E
TPU	Unit0 (TPU0 to TPU5)	16-Bit Timer Puls Unit (TPU) Unit 0 (Channlel 0 to Channel 5)
	Unit1 (TPU6 to TPU11)	16-Bit Timer Puls Unit (TPU) Unit 1 (Channlel 6 to Channel 11)
TMR	Unit0 (TMR0 and TMR1)	8-Bit Timer (TMR) Unit 0 (Channlel 0 and 1)
	Unit1 (TMR2 and TMR3)	8-Bit Timer (TMR) Unit 1 (Channlel 2 and 3)
CMT	Unit0 (CMT0 and CMT1)	Compare Match Timer (CMT) Unit 0 (Channlel 0 and 1)
	Unit1 (CMT2 and CMT3)	Compare Match Timer (CMT) Unit 1 (Channlel 2 and 3)
SCI	SCI0 to SCI6	Serial Communications Interface (SCI) Channel 0 to 6
RIIC	RIIC0 and RIIC1	I2C Bus Interface (RIIC) Channel 0 and 1
A/D	AD0 to AD3	A/D Converter Unit0 to Unit3

For how to make the setting of peripheral modules, refer to the user’s manual. For pin function settings, refer to 3.2 Pin Functions.

3.2 Pin Functions

The pin function window opens by selecting [SYSTEM] on the peripheral-module selection tabs and selecting [Pin] on the resource pane.

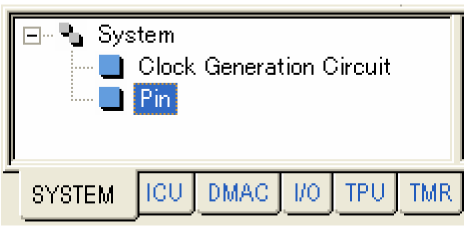
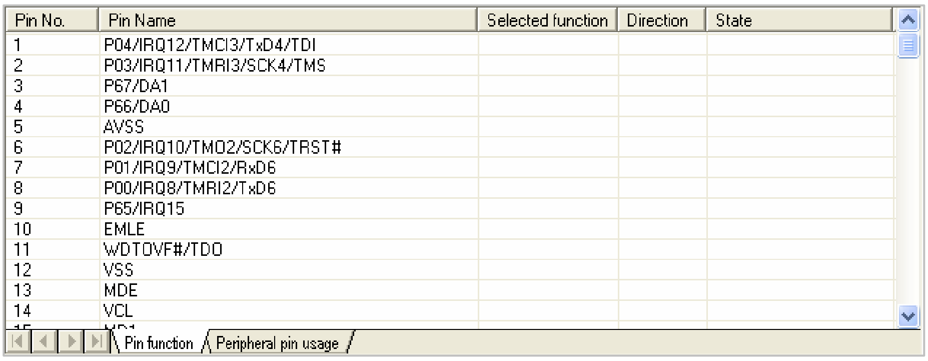


Figure 3.2 Selection to open the pin function window

The pin function window consists of [Pin function] sheet and [Peripheral pin usage] sheet.

3.2.1 Pin Function Sheet

In the pin function sheet, all pins are displayed in numerical order.



Pin No.	Pin Name	Selected function	Direction	State
1	P04/IRQ12/TMCI3/TxD4/TDI			
2	P03/IRQ11/TMRI3/SCK4/TMS			
3	P67/DA1			
4	P66/DA0			
5	AVSS			
6	P02/IRQ10/TMO2/SCK6/TRST#			
7	P01/IRQ9/TMCI2/RxD6			
8	P00/IRQ8/TMRI2/TxD6			
9	P65/IRQ15			
10	EMLE			
11	WDTOVF#/TDO			
12	VSS			
13	MDE			
14	VCL			

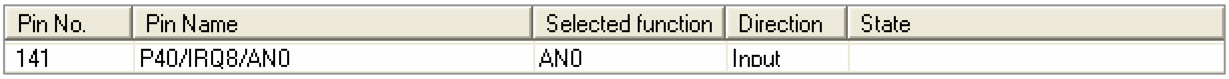
Figure 3.3 Pin function sheet

The contents of each column are shown in table 3.2.

Table 3.2 The contents of each column in the pin function sheet

Column	Contents
Pin No.	Pin number
Pin name	The name of the pin (All pin functions assigned to a pin)
Selected function	The pin function selected by the peripheral module settings
Direction	The direction (Input/Output) of the selected pin function
State	State of the setting

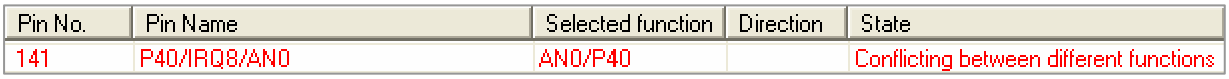
When a setting of peripheral module which uses pins is made, the result of setting is displayed in the pin function sheet. For example, if AD0 is set to convert the analog input signal of AN0 in A/D converter setting, no. the line of 141 pin which the AN0 is assigned to is displayed as shown in figure 3.4.



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

Figre 3.4 Display of selected pin function

In this state, if an I/O port P40 is set up, the confliction will be indicated as shown in figure 3.5.



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

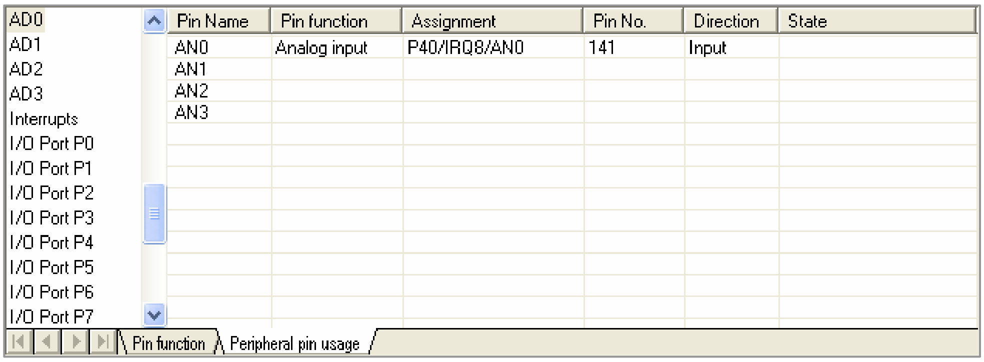
Figre 3.5 Display of confliction (Pin function sheet)

Note

- In the RX610 group, the pin function can not be selected for a pin. The pin function is determined by the settings of the peripheral modules. The pin function cannot be changed in the Pin Function sheet.
- For some pin functions, it is possible to change the pin to which the function is assigned. The pin function assignment can be changed in the Peripheral Pin Usage sheet.
- If the multiple output pin functions are enabled in one pin, the output pin function of the highest priority will be active. For details, refer to the RX610 group hardware manual.

3.2.2 Peripheral Pin Usage Sheet

The peripheral pin usage sheet shows the usage of pin functions of each peripheral module. The pin functions of the peripheral module selected in left pane are displayed in right pane.



AD0
AD1
AD2
AD3
Interrupts
I/O Port P0
I/O Port P1
I/O Port P2
I/O Port P3
I/O Port P4
I/O Port P5
I/O Port P6
I/O Port P7

Pin Name	Pin function	Assignment	Pin No.	Direction	State
AN0	Analog input	P40/IRQ8/AN0	141	Input	
AN1					
AN2					
AN3					

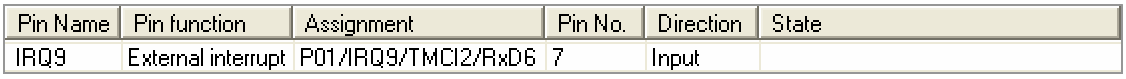
Figure 3.6 Peripheral pin usage sheet

The contents of each column are shown in table 3.3.

Table 3.2 The contents of each column in the peripheral pin usage sheet

Column	Contents
Pin Name	The pin functions of peripheral module selected in left pane
Pin Function	The usage of pin function
Assignment	The name of pin to which the pin function is assigned
Pin No.	Pin number
Direction	The direction (Input/Output)
State	State of the setting

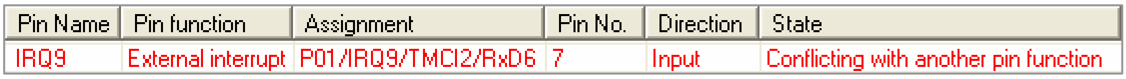
When a setting of peripheral module which uses pins is made, the result of setting is displayed in the peripheral pin usage sheet. For example, if the IRQ9 is enabled in the external interrupt setting, the line of IRQ9 is displayed as shown in figure 3.7.



Pin Name	Pin function	Assignment	Pin No.	Direction	State
IRQ9	External interrupt	P01/IRQ9/TMCI2/RxD6	7	Input	

Figre 3.7 Display of pin usage

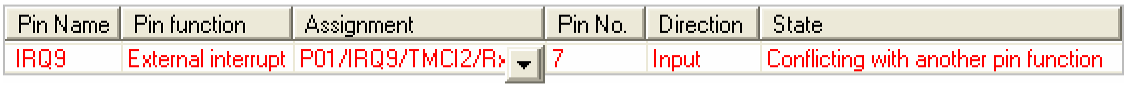
In this state, if an I/O port P01 is set up, the confliction will be indicated as shown in figure 3.8.



Pin Name	Pin function	Assignment	Pin No.	Direction	State
IRQ9	External interrupt	P01/IRQ9/TMCI2/RxD6	7	Input	Conflicting with another pin function

Figre 3.8 Display of confliction (Peripheral pin usage sheet)

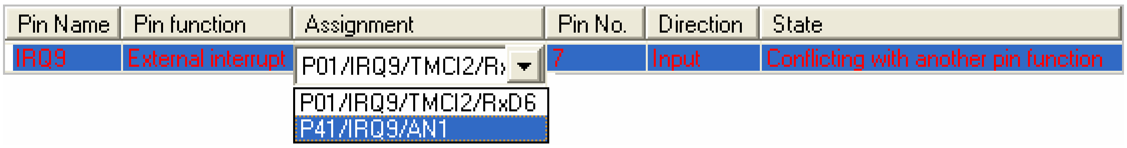
It is possible to change the pin to which the IRQ9 is assigned. To change assignment of pin function, put the mouse pointer on the Assignment cell. The drop down button to open the assignment selection opens.



Pin Name	Pin function	Assignment	Pin No.	Direction	State
IRQ9	External interrupt	P01/IRQ9/TMCI2/R›	7	Input	Conflicting with another pin function

Figre 3.9 Display of drop down button

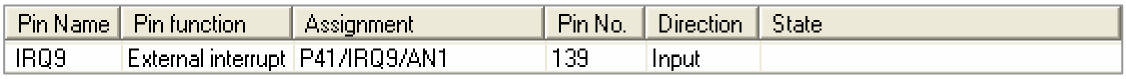
Click the drop down button and select P41/IRQ9/AN1 from the drop down menu.



Pin Name	Pin function	Assignment	Pin No.	Direction	State
IRQ9	External interrupt	P01/IRQ9/TMCI2/R›	7	Input	Conflicting with another pin function

Figre 3.10 Display of drop down menu

If P41/IRQ9/AN1 is not used by other peripheral modules, the confliction can be solved.



Pin Name	Pin function	Assignment	Pin No.	Direction	State
IRQ9	External interrupt	P41/IRQ9/AN1	139	Input	

Figre 3.11 Display of pin usage (After changing the assignment)

The pin functions of which the assignment can be changed are shown in table 3.4.

Table 3.4 The pin functions of which the assignment can be changed (RX610 144pin)
(Upper row is default)

Peripheral module	Pin function	Selection of assignment	Pin No.
TPU Unit0 (TPU0 to TPU5)	TCLKA *1	P32/IRQ2/PO10/TIOCC0/TCLKA	27
		P14/IRQ4/TCLKA/SDA1	43
	TCLKB *1	P33/IRQ3/PO11/TIOCC0/TIOCD0/TCLKB	26
		P15/IRQ5/TCLKB/SCK3/SCL1	42
	TCLKC *1	P35/PO13/TIOCA1/TIOCB1/TCLKC	40
		P16/IRQ6/TCLKC/RxD3/SDA0	50
	TCLKD *1	P37/PO15/TIOCA2/TIOCB2/TCLKD	38
		P17/IRQ7/TCLKD/TxD3/SCL0/ADTRG1#	48
TPU0	TIOCA0(IC) *2	P30/IRQ0/PO8/TIOCA0	29
		P31/IRQ1/PO9/TIOCA0/TIOCB0	28
	TIOCC0(IC) *2	P32/IRQ2/PO10/TIOCC0/TCLKA	27
		P33/IRQ3/PO11/TIOCC0/TIOCD0/TCLKB	26
TPU1	TIOCA1(IC) *2	P34/IRQ4/PO12/TIOCA1	25
		P35/PO13/TIOCA1/TIOCB1/TCLKC	50
TPU2	TIOCA2(IC) *2	P36/PO14/TIOCA2	49
		P37/PO15/TIOCA2/TIOCB2/TCLKD	48
TPU3	TIOCA3(IC) *2	P21/PO1/TIOCA3/TMCI0/RxD0	36
		P20/PO0/TIOCA3/TIOCB3/TMRI0/TxD0	37
	TIOCC3(IC) *2	P22/PO2/TIOCC3/TMO0/SCK0	35
		P23/PO3/TIOCC3/TIOCD3	34
TPU4	TIOCA4(IC) *2	P25/PO5/TIOCA4/TMCI1/RxD1	32
		P24/PO4/TIOCA4/TIOCB4/TMRI1	33
TPU5	TIOCA5(IC) *2	P26/PO6/TIOCA5/TMO1/TxD1	31
		P27/PO7/TIOCA5/TIOCB5/SCK1	30

Peripheral module	Pin function	Selection of assignment	Pin No.
TPU6	TIOCA6(IC) *2	PA0/A0/BC0#/PO16/TIOCA6	101
		PA1/A1/PO17/TIOCA6/TIOCB6	100
	TIOCC6(IC) *2	PA2/A2/PO18/TIOCC6/TCLKE	99
		PA3/A3/PO19/TIOCC6/TIOCD6/TCLKF	98
TPU7	TIOCA7(IC) *2	PA4/A4/PO20/TIOCA7	97
		PA5/A5/PO21/TIOCA7/TIOCB7/TCLKG	96
TPU8	TIOCA8(IC) *2	PA6/A6/PO22/TIOCA8	95
		PA7/A7/PO23/TIOCA8/TIOCB8/TCLKH	94
TPU9	TIOCA9(IC) *2	PB0/A8/PO24/TIOCA9	92
		PB1/A9PO25/TIOCA9/TIOCB9	85
	TIOCC9(IC) *2	PB2/A10/PO26/TIOCC9	84
		PB3/A11/PO27/TIOCC9/TIOCD9	83
TPU10	TIOCA10(IC) *2	PB4/A12/PO28/TIOCA10	82
		PB5/A13/PO29/TIOCA10/TIOCB10	81
TPU11	TIOCA11(IC) *2	PB6/A14/PO30/TIOCA11	80
		PB7/A15/PO31/TIOCA11/TIOCB11	79
ICU (External Interrupts)	IRQ0	P30/IRQ0/PO8/TIOCA0	29
		P10/IRQ0	47
	IRQ1	P31/IRQ1/PO9/TIOCA0/TIOCB0	28
		P11/IRQ1/SCK2	46
	IRQ2	P32/IRQ2/PO10/TIOCC0/TCLKA	27
		P12/IRQ2/RxD2	45
	IRQ3	P33/IRQ3/PO11/TIOCC0/TIOCD0/TCLKB	26
		P13/IRQ3/TxD2/ADTRG0#	44
	IRQ4	P34/IRQ4/PO12/TIOCA1	25
		P14/IRQ4/TCLKA/SDA1	43
	IRQ5	PE5/IRQ5/D13	104
		P15/IRQ5/TCLKB/SCK3/SCL1	42
	IRQ6	PE6/IRQ6/D14	103
		P16/IRQ6/TCLKC/RxD3/SDA0	40
	IRQ7	PE7/IRQ7/D15	102
		P17/IRQ7/TCLKD/TxD3/SCL0/ADTRG1#	38
	IRQ8	P00/IRQ8/TMRI2/TxD6	8
		P40/IRQ8/AN0	141
	IRQ9	P01/IRQ9/TMCI2/RxD6	7
		P41/IRQ9/AN1	139
	IRQ10	P02/IRQ10/TMO2/SCK6/TRST#	6
		P42/IRQ10/AN2	138
	IRQ11	P03/IRQ11/TMRI3/SCK4/TMS	2
		P43/IRQ11/AN3	137
	IRQ12	P04/IRQ12/TMCI3/TxD4/TDI	1
		P44/IRQ12/AN4	136
	IRQ13	P05/IRQ13/TMO3/RxD4/TCK	144
		P45/IRQ13/AN5	135
	IRQ14	P76/IRQ14	67
		P46/IRQ14/AN6	134
	IRQ15	P65/IRQ15	9
		P47/IRQ15/AN7	133

*1 The settings are linked together

*2 When using as an input capture pin

4. Specification of Generated Functions

Table 4.1 shows generated functions for the RX610.

Table 4.1 Generated Functions for the RX610

Clock-generation circuit

Generated Function	Description
R_PG_Clock_Set	Set up the clocks

Interrupt controller (ICU)

Generated Function	Description
R_PG_ExtInterrupt_Set_*〈interrupt type〉*	Set up an external interrupt
R_PG_ExtInterrupt_Disable_*〈interrupt type〉*	Disable the setting of an external interrupt
R_PG_ExtInterrupt_GetRequestFlag_*〈interrupt type〉*	Get an external interrupt request flag
R_PG_ExtInterrupt_ClearRequestFlag_*〈interrupt type〉*	Clear an external interrupt request flag
R_PG_FastInterrupt_Set	Set an interrupt as the fast interrupt
R_PG_Exception_Set	Set exception handlers

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

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*>	Have 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*>	Stop the data transfer
R_PG_DMAC_GetTransferredByteCount_C<*channel number*>	Get the current transfer data size
R_PG_DMAC_ClearTransferEndFlag_C<*channel number*>	Clear the transfer end flag
R_PG_DMAC_SetReload_SrcAddress_C<*channel number*>	Set the source address reload value
R_PG_DMAC_SetReload_DestAddress_C<*channel number*>	Set the destination address reload value
R_PG_DMAC_SetReload_ByteCount_C<*channel number*>	Set the transfer data size reload value
R_PG_DMAC_StopModule	Shut down the all channels of DMAC

(e) 16-Bit Timer Pulse Unit (TPU)

Generated Function	Description
R_PG_Timer_Start_TPU_U<*unit number*>_C<*channel number*>	Set up the TPU and start the count
R_PG_Timer_HaltCount_TPU_U<*unit number*>_C<*channel number*>	Halt the TPU count
R_PG_Timer_ResumeCount_TPU_U<*unit number*>_C<*channel number*>	Resume the TPU count
R_PG_Timer_GetCounterValue_TPU_U<*unit number*>_C<*channel number*>	Acquire the TPU counter value

R_PG_Timer_SetCounterValue_TPU_U*<unit number>*_C*<channel number>*	Set the TPU counter value
R_PG_Timer_GetRequestFlag_TPU_U*<unit number>*_C*<channel number>*	Acquire and clear the TPU interrupt flags
R_PG_Timer_StopModule_TPU_U*<unit number>*	Shut down the TPU unit

(d) 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〉*	Stop a TMR unit

(e) 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
R_PG_Timer_HaltCount_CMT_U*〈unit number〉*_C*〈channel number〉*	Halt the CMT count
R_PG_Timer_ResumeCount_CMT_U*〈unit number〉*_C*〈channel number〉*	Resume the CMT count
R_PG_Timer_GetCounterValue_CMT_U*〈unit number〉*_C*〈channel number〉*	Acquire the CMT counter value
R_PG_Timer_SetCounterValue_CMT_U*〈unit number〉*_C*〈channel number〉*	Set the CMT counter value
R_PG_Timer_StopModule _CMT_U*〈unit number〉*	Shut down the CMT unit

(e) Serial Communications Interface (SCI)

Generated Function	Description
R_PG_SCI_Set_C*〈channel number〉*	Set a SCI channel
R_PG_SCI_StartSending_C*〈channel number〉*	Start the data transmission
R_PG_SCI_SendAllData_C*〈channel number〉*	Transmit all data
R_PG_SCI_GetSentDataCount_C*〈channel number〉*	Acquire the number of transmitted data
R_PG_SCI_StartReceiving_C*〈channel number〉*	Start the data reception
R_PG_SCI_ReceiveAllData_C*〈channel number〉*	Receive all data
R_PG_SCI_StopCommunication_C*〈channel number〉*	Stop transmission and reception
R_PG_SCI_GetReceivedDataCount_C*〈channel number〉*	Acquire the number of received data
R_PG_SCI_GetReceptionErrorFlag_C*〈channel number〉*	Get the serial reception error flag
R_PG_SCI_GetTransmitStatus_C*〈channel number〉*	Get the state of transmission
R_PG_SCI_StopModule_C*〈channel number〉*	Shut down a SCI channel

(e) 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 the stop condition
R_PG_I2C_GetBusState_C*〈channel number〉*	Get the bus state

R_PG_I2C_SlaveMonitor_C〈*channel number*〉	Slave bus monitor
R_PG_I2C_SlaveSend_C〈*channel number*〉	Slave data transmission
R_PG_I2C_GetDetectedAddress_C〈*channel number*〉	Get the detected address
R_PG_I2C_GetTR_C〈*channel number*〉	Get the transmit/receive mode
R_PG_I2C_GetEvent_C〈*channel number*〉	Get the detected event
R_PG_I2C_GetReceivedDataCount_C〈*channel number*〉	Acquires the count of transmitted data
R_PG_I2C_GetSentDataCount_C〈*channel number*〉	Acquires the count of received data
R_PG_I2C_Reset_C〈*channel number*〉	Reset the bus
R_PG_I2C_StopModule_C〈*channel number*〉	Shut down the I2C bus interface channel

(f) A/D converter

Generated Function	Description
R_PG_ADC_10_Set_AD〈*unit number*〉	Set an A/D converter
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_StopModule_AD〈*unit number*〉	Stop an A/D converter

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), and external bus clock (BCLK) from EXTAL.

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();
}
```

4.2 Interrupt Controller (ICU)

4.2.1 R_PG_ExtInterrupt_Set_*<interrupt type>*

Definition

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

RPDL function

R_INTC_CreateExtInterrupt

Details

- Enables an external interrupt (IRQ0 to IRQ15 or the NMI) and sets the input direction and input buffer for the pins to be used for the external interrupt signal. For IRQn, the pin to be used (IRQn-A/B) 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 4.11, Notes on Notification Functions.
- If a name of the interrupt notification function is not specified in the GUI, an interrupt handler will not be called even if the external interrupt is input. The state of a request flag can be acquired by calling R_PG_ExtInterrupt_GetRequestFlag_<*interrupt type*>.

Example1

A case where Irq0ExtIntFunc 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();

    While(1);
}

//IRQ0 notification function
void Irq0ExtIntFunc (void)
{
    func_irq0();     //Processing of IRQ0
}
```

Example2

A case where a name has not been specified for 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();

    While(1){
        bool flag;

        //Acquire the interrupt request flag for IRQ0.
        R_PG_ExtInterrupt_GetRequestFlag_IRQ0( &flag );
        if( flag ){
            func_irq0();     //Processing of IRQ0
        }

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

4.2.2 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 Irq0ExtIntFunc 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();

    While(1);
}

//External interrupt (IRQ0) notification function
void Irq0ExtIntFunc (void)
{
    //Disable IRQ0.
    R_PG_ExtInterrupt_Disable_IRQ0();

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

4.2.3 R_PG_ExtInterrupt_GetRequestFlag_*<interrupt type>*

Definition

bool R_PG_ExtInterrupt_GetRequestFlag_<*interrupt type*> (bool * flag)
<*interrupt type*>: IRQ0 to IRQ15 or the NMI

Description

Get an external interrupt request flag

Parameter

bool * flag	Destination for storage of the interrupt request flag

Return value

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

File for output

R_PG_ExtInterrupt_<*interrupt type*>.c
<*interrupt type*>: IRQ0 to IRQ15 or the NMI

RPDL function

R_INTC_GetExtInterruptStatus

Details

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

Example

A case where a name has not been specified for 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();

    While(1){
        bool flag;

        //Acquire the interrupt request flag for IRQ0.
        R_PG_ExtInterrupt_GetRequestFlag_IRQ0( &flag );
        if( flag ){
            func_irq0();     //Processing of IRQ0
        }

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

4.2.4 R_PG_ExtInterrupt_ClearRequestFlag_<*interrupt type*>

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

RPDL function
R_INTC_ControlExtInterrupt

Details

- Clears the interrupt request flag for an external interrupt (IRQ0 to IRQ15 or the NMI).
- This operation will not work for a level-sensitive interrupt if the input signal is still low.

Example
A case where a name has not been specified for 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();

    While(1){
        bool flag;

        //Acquire the interrupt request flag for IRQ0.
        R_PG_ExtInterrupt_GetRequestFlag_IRQ0( &flag );
        if( flag ){
            func_irq0();    //Processing of IRQ0
        }

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

4.2.5 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();
}
```

4.2.6 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 4.11, 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
}
```

4.3 I/O Ports

4.3.1 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 E

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 E

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();
}
```

4.3.2 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 E
<*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 E

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();
}
```

4.3.3 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 E

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 E

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 );
}
```

4.3.4 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 E
<*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 E)

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);
}
```

4.3.5 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 E

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 E

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_Set_P0( 0x03 );
}
```

4.3.6 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 E
<*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 E

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 );
}
```

4.4 DMAC controller (DMAC)

4.4.1 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 4.11, Notes on Notification Functions.
- To transfer the SCI transmission data by DMAC, make the following settings.

  DMAC settings

Transfer system	: Single-operand transfer
Destination start address	: Address of serial transmit data register
Address addition direction	: Fixed
Unit data size	: 1 byte
Single operand data count	: 1

  SCI setting

Data transmission method	: Transfer the transmitted serial data by DMAC

  For usage of function, refer to example 2.

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

  DMAC settings

Transfer system	: Single-operand transfer
Source start address	: Address of serial receive data register
Address addition direction	: Fixed
Unit data size	: 1 byte
Single operand data count	: 1

SCI 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 IRQ0
    R_PG_ExtInterrupt_Set_IRQ0();

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

//The notification function which is called when the transfer completes
void Dmac0IntFunc (void)
{
    //Stop the DMAC
    R_PG_DMAC_StopModule();
}
```

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

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

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

    //Set up DMAC0
    R_PG_DMAC_Set_C0();

    //Set up SCI0
    R_PG_SCI_Set_C0();

    //Have DMAC0 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);
}
```

```
//The notification function which is called when the transfer completes
void Dmac0IntFunc (void)
{
    //SCI transmit end flag
    bool sci_transfer_cmplete;
    sci_transfer_cmplete = false;

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

    //Stop the SCI
    R_PG_SCI_StopCommunication();

    //Stop the DMAC
    R_PG_DMAC_StopModule();

    sci_dma_transfer_complete = ture;
}
```

<u>Example 3</u> A case where the SCI reception data is transferred by DMAC

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

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

//DMA transfer end flag
volatile uint8_t sci_dma_transfer_complete;

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

    //Set up DMAC0
    R_PG_DMAC_Set_C0();

    //Set up SCI0
    R_PG_SCI_Set_C0();

    //Have 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
    );
}

//The notification function which is called when the transfer completes
void Dmac0IntFunc (void)
{
    //Stop the SCI reception
    R_PG_SCI_StopCommunication

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

4.4.2 R_PG_DMAC_Activate_C<*channel number*>

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

Description: Have 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
<*unit number*>: 0 to 3

RPDL function: R_DMAC_Control

Details:

- This function has the DMAC channel 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
- 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();

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

//The notification function which is called when the transfer completes
void Dmac0IntFunc (void)
{
      //Stop the DMAC
      R_PG_DMAC_StopModule();
}
```

4.4.3 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
<*unit 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
- 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();
}

//The notification function which is called when the transfer completes
void Dmac0IntFunc (void)
{
      //Stop the DMAC
      R_PG_DMAC_StopModule();
}
```

4.4.4 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
<*unit number*>: 0 to 3

RPDL function: R_DMAC_Control

Details:
- This function suspends the DMA transfer.

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
  Irq1ExtIntFunc was specified as the IRQ1 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();

     //Set IRQ1
      R_PG_ExtInterrupt_Set_IRQ1();

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

//The notification function which is called when the transfer completes
void Dmac0IntFunc (void)
{
      //Stop the DMAC
      R_PG_DMAC_StopModule();
}

//IRQ1 interrupt notification function
void Irq1ExtIntFunc (void)
{
      //Suspend the DMA transfer
      R_PG_DMAC_Suspend_C0();
}
```

4.4.5 R_PG_DMAC_GetTransferredByteCount_C*<channel number>*

Definition bool R_PG_DMAC_GetTransferredByteCount_C<*channel number*> (uint32_t * data)
< *channel number* > : 0 to 3

Description Get the current transfer byte count register value

Parameter

uint32_t * data	The address of storage area for the current transfer byte count register value

Return value

true	Acquisition succeeded
false	Acquisition failed.

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

RPDL function R_DMAC_GetStatus

Details

- This function gets the current transfer byte count register value.

Example A case where the setting is made as follows.

- The software trigger was selected as a transfer start trigger of DMAC0

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

void func(void)
{
    uint32_t count;

    //Set up DMAC0
    R_PG_DMAC_Set_C0();

    //Start the DMA transfer of DMAC0
    R_PG_DMAC_StartTransfer_C0();

    //Wait for the current transfer byte count register value to become 10
    do{
        R_PG_DMAC_GetTransferredByteCount_C0( & count );
    } while( count > 10 );

    //Suspend the DMA transfer
    R_PG_DMAC_Suspend_C0();
}
```

4.4.6 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 DMA transfer end flag

Parameter

None

Return value

true	Clearing succeeded
false	Clearing failed

File for output

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

RPDL function

R_DMAC_Control

Details

- This function clears the DMA transfer end flag.
- This flag is cleared automatically if a notification function is enabled in GUI.

Example

A case where the setting is made as follows.

- The software trigger was selected as a transfer start trigger of DMAC0
- The DMA interrupt was not enabled

```
#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();

      //Clear the DMA transfer end flag of DMAC0
      R_PG_DMAC_ClearTransferEndFlag_C0();

      //Start the DMA transfer of DMAC0
      R_PG_DMAC_StartTransfer_C0();
}
```

4.4.7 R_PG_DMAC_SetReload_SrcAddress_C<*channel number*>

Definition

bool R_PG_DMAC_SetReload_SrcAddress_C<*channel number*> (uint32_t data)
< *channel number* > : 0 to 3

Description

Set the source address reload value

Conditions for output

Enable the source address reload

Parameter

uint32_t data	The source address reload value

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_Control

Details

- This function sets the source address reload value.
- Call this function from DMA interrupt notification function.

Example

A case where the source address reload, the destination address reload, and the transfer data size reload are enabled.

- Consecutive-operand transfer is selected as a transfer system.
- IRQ0 interrupt was selected as a transfer start trigger of DMAC0
- Dmac0IntFunc was specified as the DMA interrupt notification function name
- The source address reload, the destination address reload, and the transfer data size reload are enabled.

```
#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();

    // Have DMAC0 ready for the transfer start trigger
    R_PG_DMAC_Activate_C0();
}
//The notification function which is called when the transfer completes
void Dmac0IntFunc (void)
{
    if( contimue ){      //Reload and continue
        R_PG_DMAC_SetReload_SrcAddress_C0( src_address );   //Source address reload
        R_PG_DMAC_SetReload_DestAddress_C0( dest_address );   //Destination address reload
        R_PG_DMAC_SetReload_ByteCount_C0( byte_count );   //Transfer data size reload
    }
    else{      //Stop the DMAC0
        R_PG_DMAC_Suspend_C0();
    }
}
```

4.4.8 R_PG_DMAC_SetReload_DestAddress_C<*channel number*>

Definition

bool R_PG_DMAC_SetReload_DestAddress_C<*channel number*> (uint32_t data)
< *channel number* > : 0 to 3

Description

Set the destination address reload value

Conditions for output

Enable the destination address reload

Parameter

uint32_t data	The destination address reload value

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_Control

Details

- This function sets the destination address reload value.
- Call this function from DMA interrupt notification function.

Example

A case where the source address reload, the destination address reload, and the transfer data size reload are enabled.

- Consecutive-operand transfer is selected as a transfer system.
- IRQ0 interrupt was selected as a transfer start trigger of DMAC0
- Dmac0IntFunc was specified as the DMA interrupt notification function name
- The source address reload, the destination address reload, and the transfer data size reload are enabled.

```
#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();

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

//The notification function which is called when the transfer completes
void Dmac0IntFunc (void)
{
    if( contimue ){     //Reload and continue
        R_PG_DMAC_SetReload_SrcAddress_C0( src_address );   //Source address reload
        R_PG_DMAC_SetReload_DestAddress_C0( dest_address );   //Destination address reload
        R_PG_DMAC_SetReload_ByteCount_C0( byte_count );   //Transfer data size reload
    }
    else{     //Stop the DMAC0
        R_PG_DMAC_Suspend_C0();
    }
}
```

4.4.9 R_PG_DMAC_SetReload_ByteCount_C<*channel number*>

Definition

bool R_PG_DMAC_SetReload_ByteCount_C<*channel number*> (uint32_t data)

< *channel number* > : 0 to 3

Description

Set the transfer data size reload value

Conditions for output

Enable the transfer data size reload

Parameter

uint32_t data	The transfer data size reload value

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_Control

Details

- This function sets the transfer data size reload value.
- Call this function from DMA interrupt notification function.

Example

A case where the source address reload, the destination address reload, and the transfer data size reload are enabled.

- Consecutive-operand transfer is selected as a transfer system.
- IRQ0 interrupt was selected as a transfer start trigger of DMAC0
- Dmac0IntFunc was specified as the DMA interrupt notification function name
- The source address reload, the destination address reload, and the transfer data size reload are enabled.

```
#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();

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

//The notification function which is called when the transfer completes
void Dmac0IntFunc (void)
{
    if( contimue ){     //Reload and continue

        R_PG_DMAC_SetReload_SrcAddress_C0( src_address );     //Source address reload
        R_PG_DMAC_SetReload_DestAddress_C0( dest_address );     //Destination address reload
        R_PG_DMAC_SetReload_ByteCount_C0( byte_count );     //Transfer data size reload
    }
    else{     //Stop the DMAC0
        R_PG_DMAC_Suspend_C0();
    }
}
```

4.4.10 R_PG_DMAC_StopModule

Definition bool R_PG_DMAC_StopModule (void)

Description Shut down the all channels of DMAC

Parameter None

Return value

true	Shutting down succeeded.
false	Shutting down failed.

File for output R_PG_DMAC.c

RPDL function R_DMAC_Destroy

Details

- Stops the all DMAC channels and places it in the module-stop state.
- Call To R_PG_DMAC_Suspend_C<*channel number*> to stop a single channel.

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();
}

//The notification function which is called when the transfer completes
void Dmac0IntFunc (void)
{
      //Stop the DMAC
      R_PG_DMAC_StopModule();
}
```

4.5 16-Bit Timer Pulse Unit (TPU)

4.5.1 R_PG_Timer_Start_TPU_U<unit number>_C<channel number>

Definition

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

Description

Set up the TPU and start the count

Parameter

None

Return value

true	Setting was made correctly.
false	Setting failed.

File for output

R_PG_Timer_TPU_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0 and 1
<*channel number*>: 0 to 11

RPDL function

R_TPU_Create

Details

- Releases the TPU from the module-stop, makes initial settings, and starts the TPU counting.
- Interrupts of the TPU 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 4.11, Notes on Notification Functions.
- If a name for the interrupt notification function is not specified in the GUI, an interrupt handler will not be called even if the interrupt occurs. The state of a request flag can be acquired by calling R_PG_Timer_GetRequestFlag_TPU_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.

Example

A case where the setting is made as follows.

- TPU unit 1 channel 6 was set up
- Tpu6IcCmAIntFunc 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_Start_TPU_U1_C6();      //Set up the TPU6 and start count
}

void Tpu6IcCmAIntFunc(void)
{
     func_cmA();      //Processing in response to a compare match A interrupt
}
```

4.5.2 R_PG_Timer_HaltCount_TPU<unit number>_C<channel number>

Definition

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

Description

Halt the TPU count

Parameter

None

Return value

true	Halting succeeded.
false	Halting failed.

File for output

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

RPDL function

R_TPU_Control

Details

- Halts counting by a TPU. To make the TPU resume counting, call the following function.
  R_PG_Timer_ResumeCount_TPU_U<*unit number*>_C<*channel number*>

Example

A case where the setting is made as follows.

- TPU unit 1 channel 6 was set up
- Tpu6IcCmAIntFunc 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_Start_TPU_U1_C6();     //Set up the TPU6 and start count
}

void Tpu6IcCmAIntFunc(void)
{
    R_PG_Timer_HaltCount_TPU_U1_C6();     //Halt the TPU6 count

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

    R_PG_Timer_ResumeCount_TPU_U1_C6();     //Resume the TPU6 count
}
```

4.5.3 R_PG_Timer_ResumeCount_TPU_U<*unit number*>_C<*channel number*>

Definition

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

Description

Resume the TPU count

Parameter

None

Return value

true	Resuming count succeeded.
false	Resuming count failed.

File for output

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

RPDL function

R_TPU_Control

Details

- Resumes counting by a TPU that was halted by R_PG_Timer_HaltCount_TPU_U<*unit number*>_C<*channel number*>.

Example

A case where the setting is made as follows.

- TPU unit 1 channel 6 was set up
- Tpu6IcCmAIntFunc 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_Start_TPU_U1_C6();      //Set up the TPU6 and start count
}

void Tpu6IcCmAIntFunc(void)
{
    R_PG_Timer_HaltCount_TPU_U1_C6();      //Halt the TPU6 count

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

    R_PG_Timer_ResumeCount_TPU_U1_C6();      //Resume the TPU6 count
}
```

4.5.4 R_PG_Timer_GetCounterValue_TPU_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_GetCounterValue_TPU_U<*unit number*>_C<*channel number*>
(uint16_t * data)
<*unit number*>: 0 or 1
<*channel number*>: 0 to 11

Description

Acquire the TPU 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_TPU_U<*unit number*>_C<*channel number*>.c
<*unit number*>: 0 or 1
<*channel number*>: 0 to 11

RPDL function

R_TPU_Read

Details

- Acquires the counter value of a TPU.

Example

A case where the setting is made as follows.

- TPU unit 0 channel 0 was set up
- Set TGRA as an input capture register and enable an input capture interrupt
- Tpu0IcCmAIntFunc 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;

void func(void)
{
    R_PG_Timer_Start_TPU_U0_C0();    //Set up the TPU0 and start count
}

void Tpu0IcCmAIntFunc(void)
{
    // Acquire the value of a TPU0 counter
    R_PG_Timer_GetCounterValue_TPU_U0_C0( &counter );
}
```

4.5.5 R_PG_Timer_SetCounterValue_TPU_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_SetCounterValue_TPU_U<*unit number*>_C<*channel number*>
(uint16_t data)
<*unit number*>: 0 or 1
<*channel number*>: 0 to 11

Description

Set the TPU counter value

Parameter

uint16_t data	Value to be set to the counter

Return value

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

File for output

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

RPDL function

R_TPU_Control

Details

- Set the counter value of a TPU.

Example

A case where the setting is made as follows.

- TPU unit 0 channel 1 was set up
- Set TGRA as an output compare register and enable a compare match interrupt
- Tpu1IcCmAIntFunc 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 func1(void)
{
    R_PG_Timer_Start_TPU_U0_C1();       //Set up the TPU1 and start count
}

void Tpu1IcCmAIntFunc(void)
{
    R_PG_Timer_SetCounterValue_TPU_U0_C1( 0 );      // Set the value of a TPU1
counter
}
```

4.5.6 R_PG_Timer_GetRequestFlag_TPU_U<*unit number*>_C<*channel number*>

Definition

```
bool R_PG_Timer_GetRequestFlag_TPU_U<unit number>_C<channel number> (
      bool* a,
      bool* b,
      bool* c,
      bool* d,
      bool* ov,
      bool* un
);
```

<*unit number*>: 0 or 1

<*channel number*>: 0 to 11

Description

Acquire and clear the TPU interrupt flags

Parameter

bool* a	The address of storage area for the compare match/input capture A flag
bool* b	The address of storage area for the compare match/input capture B flag
bool* c	The address of storage area for the compare match/input capture C flag
bool* d	The address of storage area for the compare match/input capture D flag
bool* ov	The address of storage area for the overflow flag
bool* un	The address of storage area for the underflow flag

Return value

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

File for output

R_PG_Timer_TPU_U<*unit number*>.c

<*unit number*>: 0 or 1

<*channel number*>: 0 to 11

RPDL function

R_TPU_Read

Details

- This function acquires the interrupt flags of TPU.
- 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, 6, and 9. Specify 0 for other channels.

Example

A case where the setting is made as follows.

- TPU unit 0 channel 1 was set up
- Set TGRA as an output compare register and enable an output compare interrupt

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

uint16_t counter;

void func(void)
{
    R_PG_Timer_Start_TPU_U0_C1();      //Set up the TPU1 and start count

    //Wait for the compare match A
    do{
        R_PG_Timer_GetRequestFlag_TPU_U0_C1(
            & cma_flag,
            0,
            0,
            0,
            0,
            0
        );
    } while( !cma_flag );

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

    // Stop the TPU unit 0
    R_PG_Timer_StopModule_TPU_U0( &counter );
}
```

4.5.7 R_PG_Timer_StopModule_TPU_U<*unit number*>

Definition

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

Description

Shut down the TPU unit

Parameter

None

Return value

true	Shutting down succeeded.
false	Shutting down failed.

File for output

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

RPDL function

R_TPU_Destroy

Details

- Stops a TPU 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.
  R_PG_Timer_HaltCount_TPU_U<*unit number*>_C<*channel number*>

Example

A case where the setting is made as follows.

- TPU unit 0 channel 1 was set up
- Tpu1IcCmAIntFunc 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.

uint16_t counter;

void func(void)
{
    R_PG_Timer_Start_TPU_U0_C1();       //Set up the TPU1 and start count
}

void Tpu1IcCmAIntFunc(void)
{
    // Stop the TPU unit 0
    R_PG_Timer_StopModule_TPU_U0( &counter );
}
```

4.6 8-Bit Timer (TMR)

4.6.1 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

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 4.11, Notes on Notification Functions.
  If a name for the interrupt notification function is not specified in the GUI, an interrupt
- handler will not be called even if the interrupt occurs. 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
        }
    }
}
```

4.6.2 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

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 counting by a TMR. To make the TMR resume counting, 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();
}
```

4.6.3 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

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 counting by a TMR 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();
}
```

4.6.4 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 data;

    //Acquire the value of a counter of TMR0.
    R_PG_Timer_GetCounterValue_TMR_U0_C0( &data );

    return data;
}
```

4.6.5 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 set to the counter

Return value

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

File for output

R_PG_Timer_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 );

    return;
}
```

4.6.6 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
}
```

4.6.7 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();
}
```

4.7 Compare Match Timer (CMT)

4.7.1 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

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 4.11, 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 count
}

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

4.7.2 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

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 counting by a CMT. To make the CMT resume counting, 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 count
}

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

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

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

4.7.3 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

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 counting by a CMT 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 count
}

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

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

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

4.7.4 R_PG_Timer_GetCounterValue_CMT_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_GetCounterValue_CMT_U<*unit number*>_C<*channel number*>
(uint16_t * data)
<*unit number*>: 0 or 1
<*channel number*>: 0 to 3

Description

Acquire the CMT counter value

Parameter

uint16_t * data	Destination for storage of the counter value

Return value

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

File for output

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

RPDL function

R_CMT_Read

Details

- Acquires the counter value of a CMT.

Example

A case where the setting is made as follows.

- CMT unit 0 channel 0 was set up

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

uint16_t counter;

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

unt16_t func2(void)
{
    uint16_t data;

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

    return data;
}
```

4.7.5 R_PG_Timer_SetCounterValue_CMT_U<*unit number*>_C<*channel number*>

Definition

bool R_PG_Timer_SetCounterValue_CMT_U<*unit number*>_C<*channel number*>
(uint16_t data)
<*unit number*>: 0 or 1
<*channel number*>: 0 to 3

Description

Set the CMT counter value

Parameter

uint16_t data	Value to be set to the counter

Return value

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

File for output

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

RPDL function

R_CMT_Control

Details

- Set the counter value of a CMT.

Example

A case where the setting is made as follows.

- CMT unit 0 channel 0 was set up

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

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

void func2(void)
{
    R_PG_Timer_SetCounterValue_CMT_U0_C0( 0 ); // Set the value of a CMT0 counter

    return;
}
```

4.7.6 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 count
}

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

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

4.8 Serial Communications Interface (SCI)

4.8.1 R_PG_SCI_Set_C<*channel number*>

Definition

bool R_PG_SCI_Set_C<*channel number*> (void)
<*channel number*>: 0 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 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. This function also disables the alternative modes on those pins.
- Function R_PG_Clock_Set must be called before any use of 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 4.11, 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.
}
```

4.8.2 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 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 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 4.11, 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();
}
```

4.8.3 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 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 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
}
```

4.8.4 R_PG_SCI_GetSentDataCount_C<*channel number*>

Definition

bool R_PG_SCI_GetSentDataCount_C<*channel number*> (uint16_t * count)
<*channel number*>: 0 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 last byte by function call" is selected as the transmit end notification 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 6

RPDL function

R_SCI_GetStatus

Details

- When "Notify the transmission completion of last byte by function call" is selected as the transmit end notification in GUI, the number of transmitted data can be aquired 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_StopSending_C0();       //Terminate the transmission
     }
}
```

4.8.5 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 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 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 4.11, 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();
}
```

4.8.6 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 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 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
}
```

4.8.7 R_PG_SCI_StopCommunication_C<*channel number*>

Definition

R_PG_SCI_StopCommunication_C<*channel number*> (void)
<*channel number*>: 0 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 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
     }
}
```

4.8.8 R_PG_SCI_GetReceivedDataCount_C<*channel number*>

Definition
bool R_PG_SCI_GetReceivedDataCount_C<*channel number*> (uint16_t * count)
<*channel number*>: 0 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 6

RPDL function
R_SCI_GetStatus

Details

- When "Notify the reception completion of last byte by function call" is selected as the receive end notification in GUI, the number of received data can be aquired 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
     }
}
```

4.8.9 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 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 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.
- 1 is set to detected error flag

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 );
}
```

4.8.10 R_PG_SCI_GetTransmitStatus_C<*channel number*>

Definition

bool R_PG_SCI_GetTransmitStatus_C<*channel number*> (bool * complete)
<*channel number*>: 0 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 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*>

4.8.11 R_PG_SCI_StopModule_C<*channel number*>

Definition
bool R_PG_SCI_StopModule_C<*channel number*> (void)
<*channel number*>: 0 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 6

RPDL function
R_SCI_Destroy

Details

- Stops a SCI channel and places it in the module-stop state.

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 transmit end notification instead of specifying the transmit 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
}
```

4.9 I2C Bus Interface (RIIC)

4.9.1 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 a 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. This function also disables the alternative modes on those pins.
  Function R_PG_Clock_Set must be called before any use of this function.

Example RIIC0 has been set in the GUI.

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

void func(void)
{
      R_PG_Clock_Set();             //The clock-generation circuit has to be set first
      R_PG_I2C_Set_C0();           //Set up RIIC0
}
```

4.9.2 R_PG_I2C_MasterReceive_C<*channel number*>

Definition

bool R_PG_I2C_MasterReceive_C<*channel number*>
(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

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 receive.
  Create the notification function as follows:
  void <*name of the notification function*> (void)
  For the notification function, note the contents of 4.11, Notes on Notification Functions.
- A Start condition will be generated automatically. If the previous transfer did not issue a stop condition, a repeated start condition will be generated.
- In the 7-bit address mode, [8:1] of specified slave address value will be output. In 10-bit address mode, [10:9] and [8:0] of specified slave address will be output.
- The number of received data can be aquired by R_PG_I2C_GetReceivedDataCount_C <*channel number*>.

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(
        6,      //Slave address
        &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();
}
```

4.9.3 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 TIIC0)
  Transfer system : Single-operand transfer
  Unit data size : 1 byte
  Single operand data count : 1
  Total transfer data size : Number of dtat to be received by RIIC0
  Source start address : Address of RIIC0 received data register
  Destination start address : Destination address of the data transfer
  DMA interrupt notification fuction name : Dmac0IntFunc

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

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

    //Isse NACK and STOP condition and complete the reception
    R_PG_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
    R_PG_PG_I2C_MasterReceive_C0(
        6,     //Slave address
        &data, // The address of the storage area (For DMAC transfer, set PDL_NO_PTR)
        10     // The number of the data (For DMAC transfer, set 0)
    );
}
```

4.9.4 R_PG_I2C_MasterSend_C<*channel number*>

Definition

bool R_PG_I2C_MasterSend_C<*channel number*>
(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

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 4.11, Notes on Notification Functions.
- A Start condition will be generated automatically. If the previous transfer did not issue a stop condition, a repeated start condition will be generated.
- In the 7-bit address mode, [8:1] of specified slave address value will be output. In 10-bit address mode, [10:9] and [8:0] of specified slave address will be output.
- The number of transmitted data can be aquired by R_PG_I2C_GetSentDataCount_C <*channel number*>.

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(
        6,      //Slave address
        &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();
}
```

4.9.5 R_PG_I2C_MasterSendWithoutStop_C<*channel number*>

Definition

bool R_PG_I2C_MasterSendWithoutStop_C<*channel number*>
(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

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 4.11, Notes on Notification Functions.
- A Start condition will be generated automatically. If the previous transfer did not issue a stop condition, a repeated start condition will be generated.
- In the 7-bit address mode, [8:1] of specified slave address value will be output. In 10-bit address mode, [10:9] and [8:0] of specified slave address will be output.
- The number of transmitted data can be aquired by R_PG_I2C_GetSentDataCount_C <*channel number*>.

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(
        6,     //Slave address
        &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();
}
```

4.9.6 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_Create

Details

- This function generates a stop condition for the reception started by R_PG_I2C_MasterReceiveWithoutStop_C<*channel number*> or 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(
        6,     //Slave address
        &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();
}
```

4.9.7 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
uint8_t busy;

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

    //Set up RIIC0
    R_PG_I2C_Set_C0();

    // Wait for the I2C bus to be free
    do{
        R_PG_I2C_GetBusState_C0( & busy );
    } while( busy );

    //Master transmission
    R_PG_I2C_MasterSend_C0(
        6,      //Slave address
        &data,      // The start address of the storage area for the data to be transmitted
        10      // The number of the data to be transmitted
    );
```

4.9.8 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 4.11, 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).

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(
        &data,      // The start address of the storage area for the received data
        10      //The number of the data to be received
    );
}

void IIC0SlaveFunc (void)
{
    bool transmit, start, stop;
    bool addr0, addr1;

    //Get the detected events
    R_PG_I2C_GetEvent_C0(0, &stop, &start, 0, 0);

    //Get an access type
    R_PG_PG_I2C_GetTR_C0(&transmit);

    //Get a detected address
    R_PG_I2C_GetDetectedAddress_C0(&addr0, &addr1, 0, 0, 0, 0);

    if (start && transmit && address0) {
        R_PG_I2C_SlaveSend_C(
            iic_data_tr_0,
            10
        );
    }

    else if (start && read && address1) {
        R_PG_I2C_SlaveSend_C(
            iic_data_tr_1,
            10
        );
    }
}
```

4.9.9 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*>

4.9.10 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.
- 1 is set to detected address

Example

Refer to the example of R_PG_I2C_SlaveMonitor_C<*channel number*>

4.9.11 R_PG_I2C_GetTR_C<*channel number*>

Definition

bool R_PG_I2C_GetTR_PG_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 mode 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.
- 1 is set to detected address

Details

- This function acquires the the transmit/receive mode.

Transmit mode flag

0	Receive mode
1	Transmit mode

Example

Refer to the example of R_PG_I2C_SlaveMonitor_C<*channel number*>

4.9.12 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.
- 1 is set to detected event.

Example

Refer to the example of R_PG_I2C_SlaveMonitor_C<*channel number*>

4.9.13 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
        &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 );
}
```

4.9.14 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
        &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 );
}
```

4.9.15 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
        &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();
    }
}
```

4.9.16 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 a 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
        &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();
}
```

4.10 A/D Converter

4.10.1 R_PG_ADC_10_Set_AD<*unit number*>

Definition bool R_PG_ADC_10_Set_AD<*unit number*> (void) <*unit number*>: 0 to 4

Description Set up an A/D converter

Parameter None

Return value

true	Setting was made correctly.
false	Setting failed.

File for output R_PG_ADC_10_AD<*unit number*>.c <*unit number*>: 0 to 4

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*>.
- In this function, the clock frequency is used to set the sampling interval. When the clock-generation circuit is in the initial state after a reset, call R_PG_Clock_Set to set the clock 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 4.11, 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.
}
```

4.10.2 R_PG_ADC_10_StartConversionSW_AD<*unit number*>

Definition
bool R_PG_ADC_10_StartConversionSW_AD<*unit number*> (void)
<*unit number*>: 0 to 4

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	Triggering conversion succeeded.
false	Triggering conversion failed.

File for output
R_PG_ADC_10_AD<*unit number*>.c
<*unit number*>: 0 to 4

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();
}
```

4.10.3 R_PG_ADC_10_StopConversion_AD<*unit number*>

Definition

bool R_PG_ADC_10_StopConversion_AD<*unit number*> (void)
<*unit number*>: 0 to 4

Description

Stop A/D conversion

Parameter

None

Return value

true	Stopping conversion succeeded.
false	Stopping conversion failed.

File for output

R_PG_ADC_10_AD<*unit number*>.c
<*unit number*>: 0 to 4

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)
}
```

4.10.4 R_PG_ADC_10_GetResult_AD_AD<*unit number*>

Definition bool R_PG_ADC_10_GetResult_AD<*unit number*> (uint16_t * data)
<*unit number*>: 0 to 4

Description Get the result of A/D conversion

Parameter

uint16_t * data	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 4

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 name for the interrupt notification function has not been specified in the GUI, and A/D conversion is not completed by the time this function is called, this function waits until the end of A/D conversion before reading the result. If the A/D-conversion start trigger is not input, processing will not return from this function. If registers of the A/D converter are modified by the program, this function will be locked.

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_AD2(&data)     //Acquire the results of A/D conversion.

    data_an0 = data[0];
    data_an1 = data[1];
    data_an2 = data[2];
    data_an3 = data[3];
}
```

4.10.5 R_PG_ADC_10_StopModule_AD<*unit number*>

Definition bool R_PG_ADC_10_StopModule_AD<*unit number*> (void)
<*unit number*>: 0 to 4

Description Shut down an A/D converter

Parameter None

Return value

true	Shutting down succeeded.
false	Shutting down failed.

File for output R_PG_ADC_10_AD<*unit number*>.c
<*unit number*>: 0 to 4

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();
}
```

4.11 Notes on Notification Functions

4.11.1 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.

4.11.2 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. Source File Registration and Building Programs in HEW

Note the following about registering the generated source files in HEW and building the program.

- The startup programs are not included in the source files generated by PDG. Select "Application" as a project type when making the HEW project to generate the startup program.
- The interrupt handlers and the vector table are included in the sources files that PDG registers in HEW. To avoid the duplication of the interrupt handlers and the vector table in startup programs generated by HEW, PDG excludes intprg.c and vecttbl.c from the build when registering the source files in HEW.
- The source files "Interrupt_xxx.c"that includes interrupt handlers are overwritten when PDG registers the source files in HEW.
- The RPDL library is produced using the default compiler options. If you specify the compiler options other than the defaults in your project, you have to utilize RPDL source under your responsibility.

6. Example of Creating an Application

This section describes a procedure for creating an application with PDG. The created sample application can work on the RSK board.

- Blink the LED on RSK with TMR interrupt
- Execute A/D conversion continuously
- Output PWM pulse with TPU
- Communicate between I2C channel 0 and channel 1

The following signs mean operation on PDG or HEW.

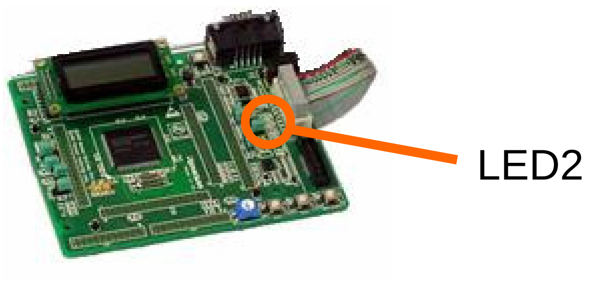
PDG : This means the operation on PDG

HEW : This means the operation on HEW

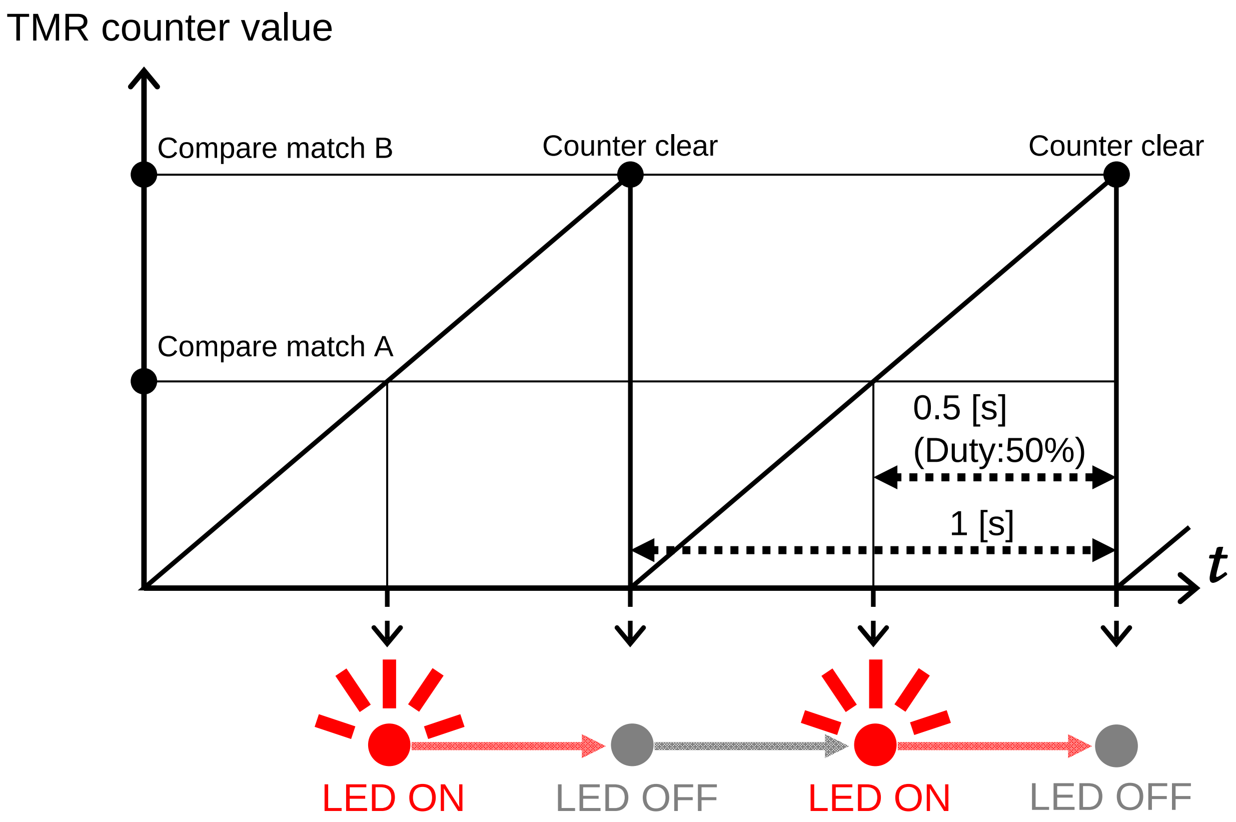
6.1 Blink the LED on RSK with TMR interrupt

The LED2 on RSK board is connected to P33. In this tutorial, 8-bit Timer and I/O port will be set up to blink this LED as follows.

The LED2 turns on when the output from P33 is 0, and turns off when the output from P33 is 1.

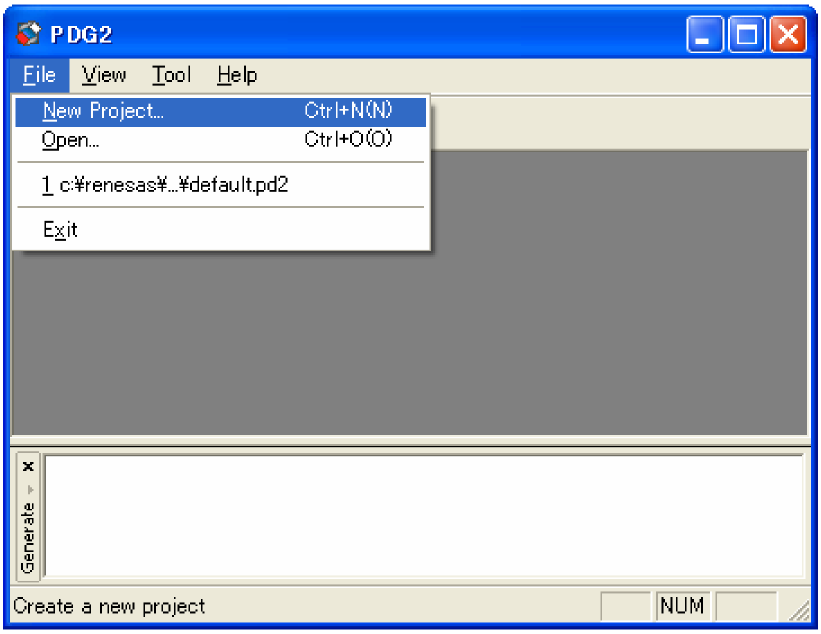


- Turn on the LED ● at compare match A
- Turn off the LED ● at compare match B
- Clear the counter at compare match B

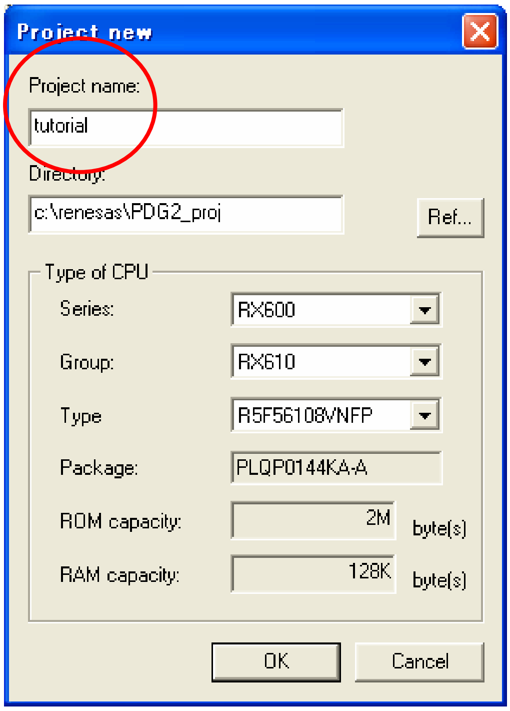


(1) Make the PDG project **PDG**

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



3. Specify "tutorial" as the project name.



Project name: tutorial

Directory: c:\renesas\PDG2_proj

Type of CPU

Series: RX600

Group: RX610

Type R5F56108VNFP

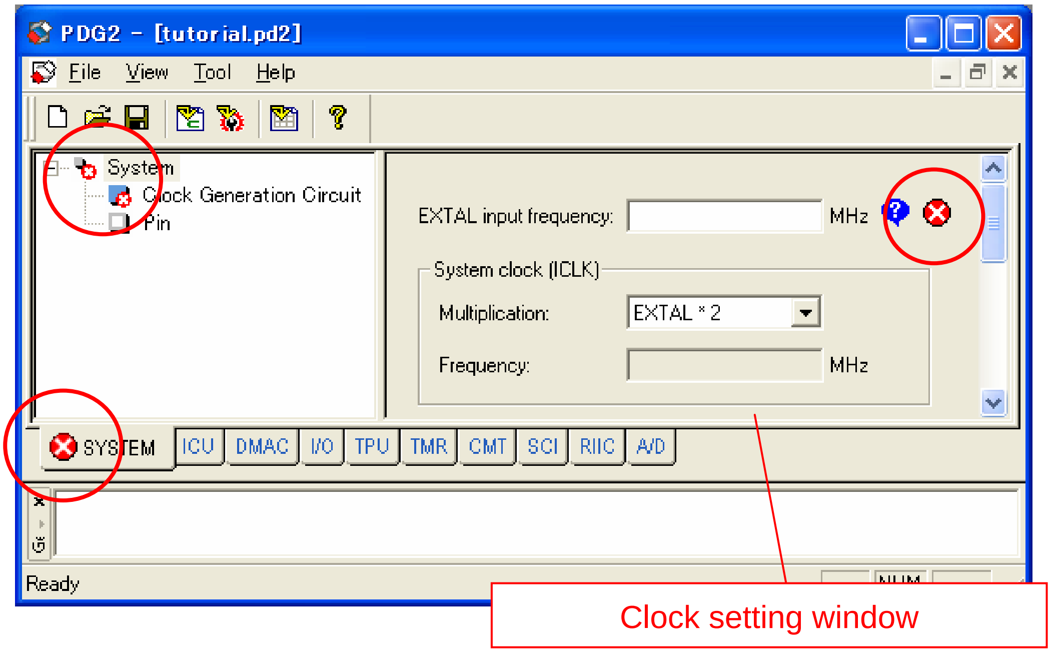
Package: PLQP0144KA-A

ROM capacity: 2M byte(s)

RAM capacity: 128K byte(s)

(2) Initial state PDG

-Immediately after making new project, clock setting window opens and an error icon is displayed.



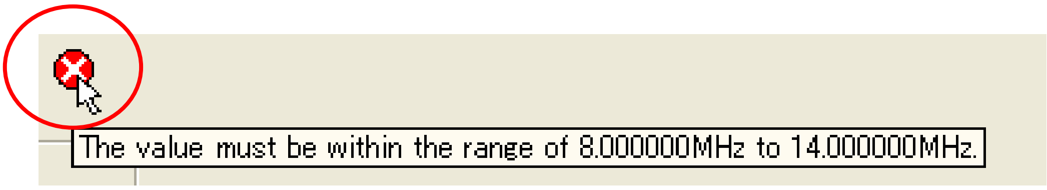
EXTAL input frequency: MHz

System clock (ICLK)

Multiplication: EXTAL * 2

Frequency: MHz

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



The value must be within the range of 8.000000MHz to 14.000000MHz.

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.

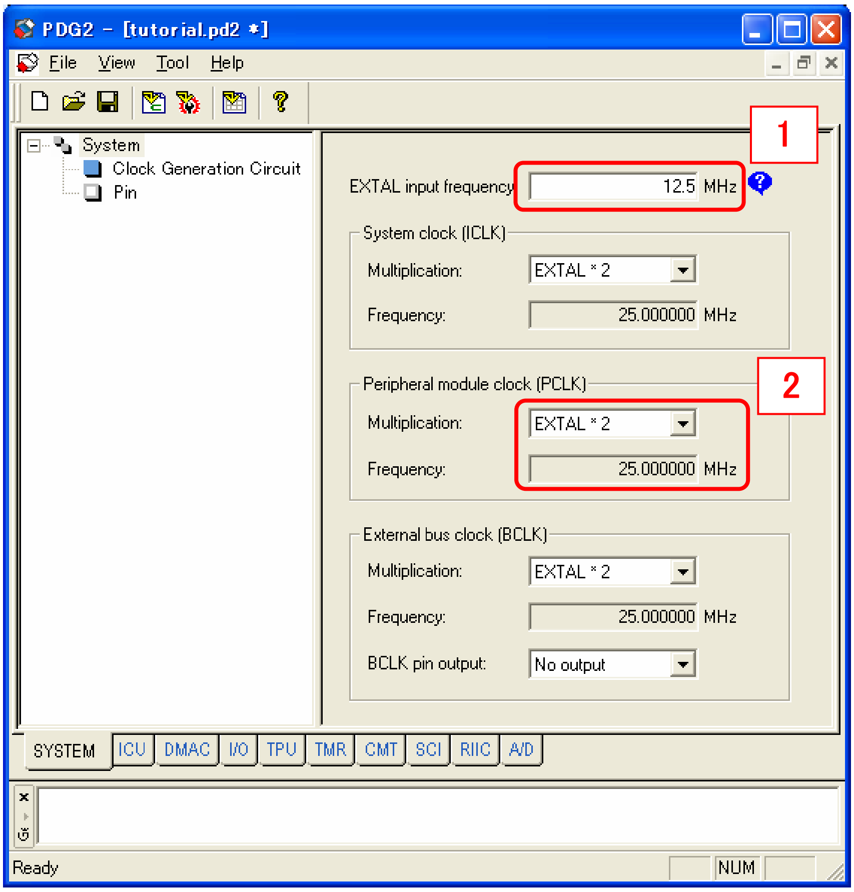
? Information

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 EXTAL clock frequency first.
   External clock frequency of the RSK board is 12.5 MHz. Set 12.5.
2. PCLK is used in 25MHz.
   Select the multiplication "EXTAL x 2" to set the PCLK to 25MHz.



EXTAL input frequency: 12.5 MHz

System clock (ICLK)
Multiplication: EXTAL * 2
Frequency: 25.000000 MHz

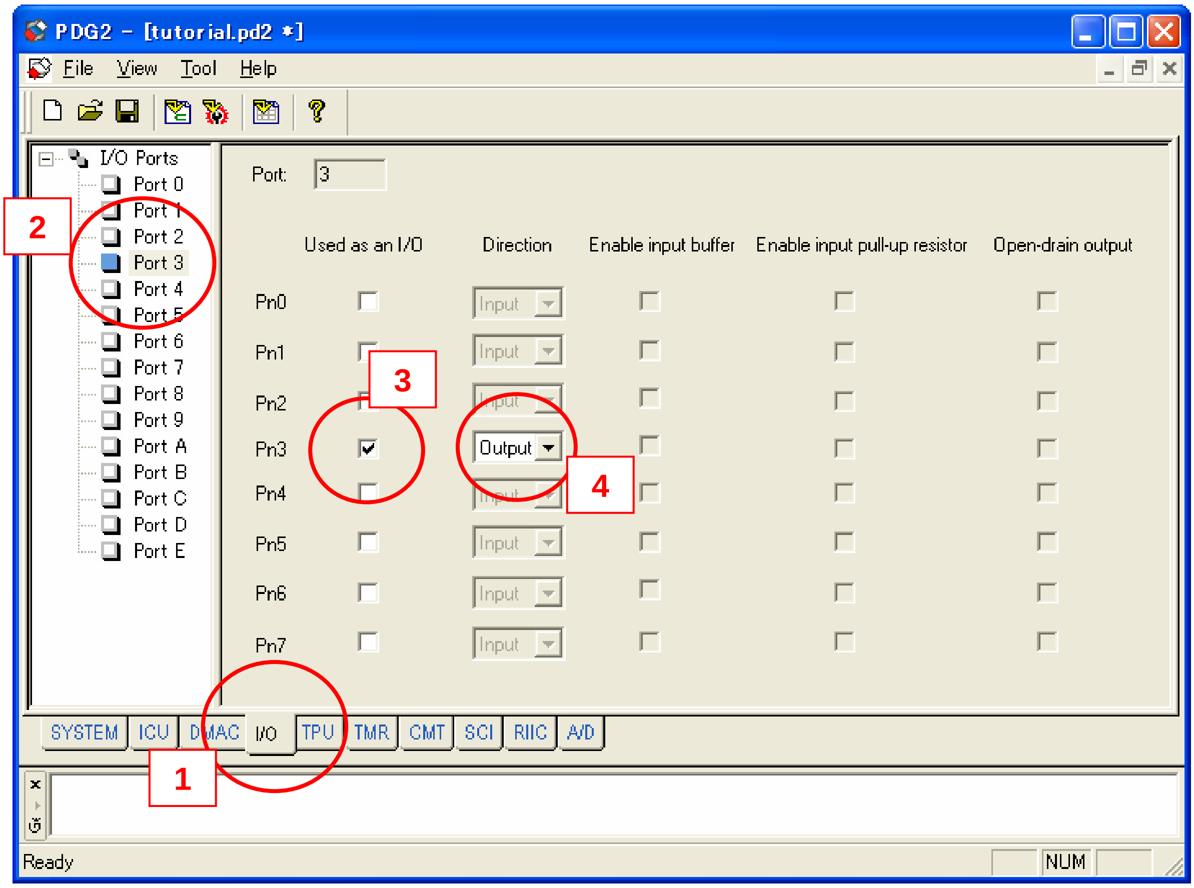
Peripheral module clock (PCLK)
Multiplication: EXTAL * 2
Frequency: 25.000000 MHz

External bus clock (BCLK)
Multiplication: EXTAL * 2
Frequency: 25.000000 MHz
BCLK pin output: No output

(4) I/O Port setting PDG

The LED1 on RSK is connected to P33 so set P33 to output port.

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



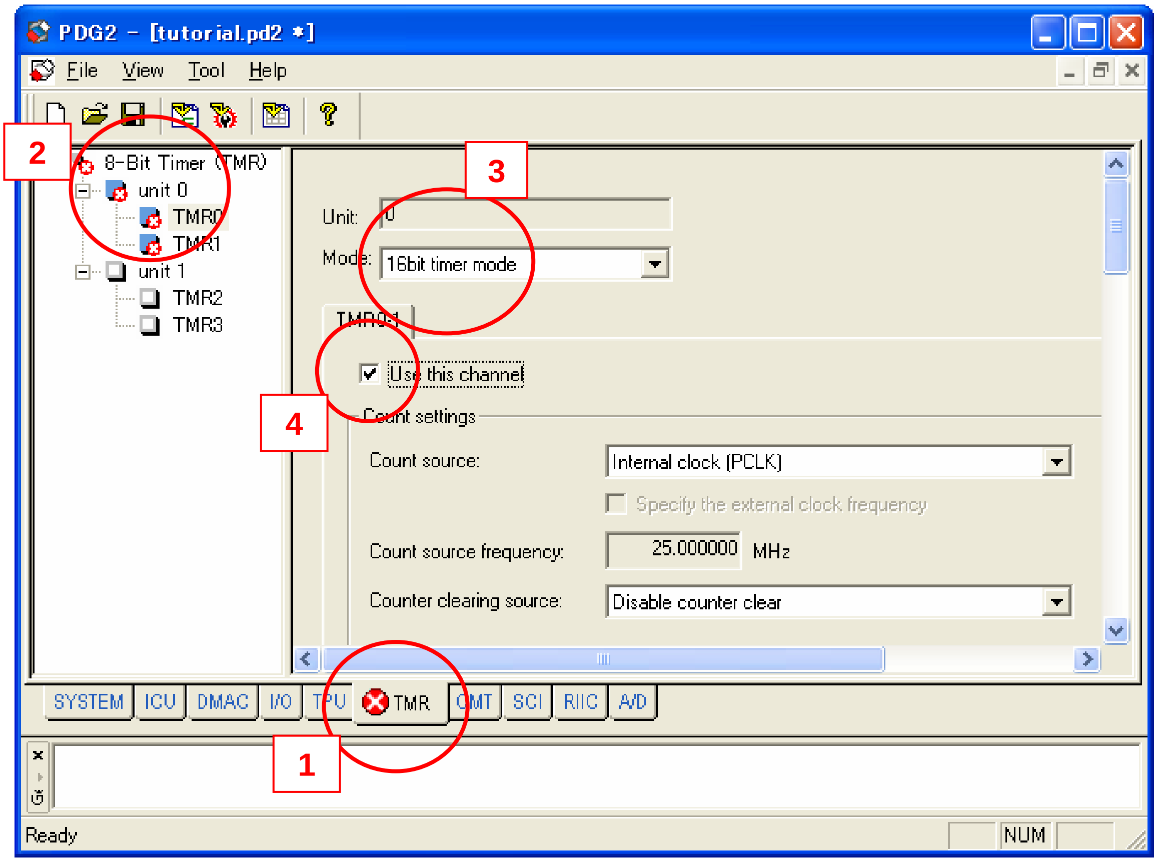
Port: 3

	Used as an I/O	Direction	Enable input buffer	Enable input pull-up resistor	Open-drain output
Pn0	☐	Input	☐	☐	☐
Pn1	☐	Input	☐	☐	☐
Pn2	☐	Input	☐	☐	☐
Pn3	☑	Output	☐	☐	☐
Pn4	☐	Input	☐	☐	☐
Pn5	☐	Input	☐	☐	☐
Pn6	☐	Input	☐	☐	☐
Pn7	☐	Input	☐	☐	☐

(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"



Unit: 0

Mode: 16bit timer mode

☑ Use this channel

Count settings

Count source: Internal clock (PCLK)

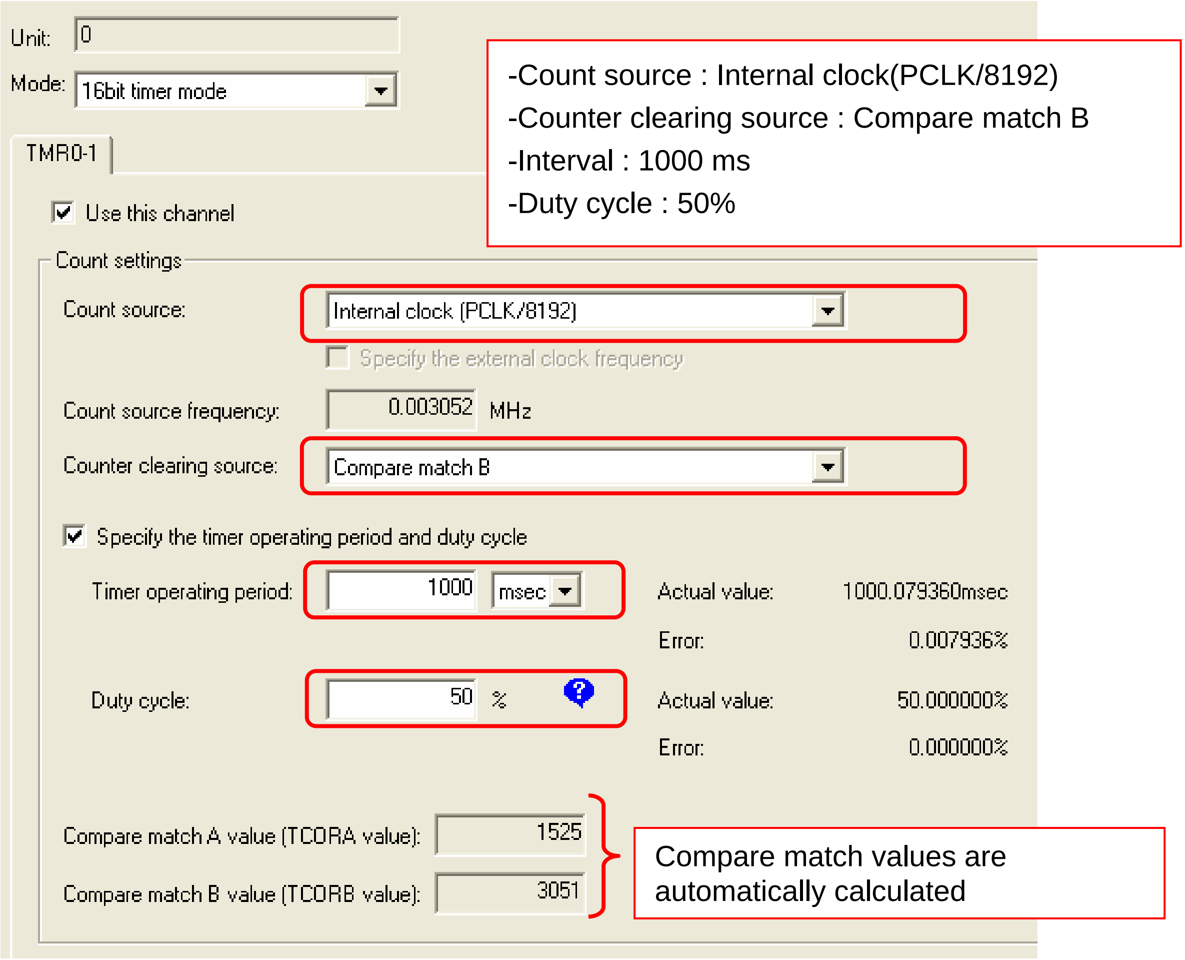
□ Specify the external clock frequency

Count source frequency: 25.000000 MHz

Counter clearing source: Disable counter clear

(6) TMR setting-2 PDG

Set the other items as follows.



Unit: 0

Mode: 16bit timer mode

TMR0-1

☑ Use this channel

Count settings

Count source: Internal clock (PCLK/8192)

□ Specify the external clock frequency

Count source frequency: 0.003052 MHz

Counter clearing source: Compare match B

☑ Specify the timer operating period and duty cycle

Timer operating period: 1000 msec — Actual value: 1000.079360msec — Error: 0.007936%

Duty cycle: 50 % — Actual value: 50.000000% — Error: 0.000000%

Compare match A value (TCORA value): 1525

Compare match B value (TCORB value): 3051

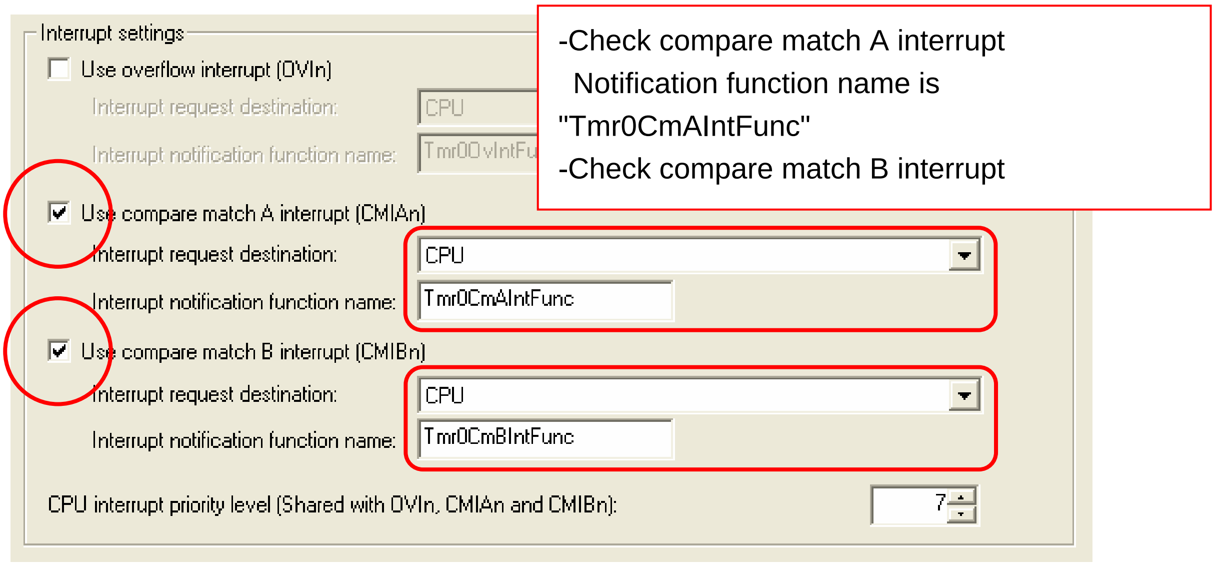
-Count source : Internal clock(PCLK/8192)
-Counter clearing source : Compare match B
-Interval : 1000 ms
-Duty cycle : 50%

Compare match values are automatically calculated

(7) TMR setting-3 PDG

Set the interrupt notification functions.

These functions are called when the interrupt occurs.



Interrupt settings

□ Use overflow interrupt (OVIn)

Interrupt request destination: CPU

Interrupt notification function name: Tmr0OvIntFu

☑ Use compare match A interrupt (CMIAn)

Interrupt request destination: CPU

Interrupt notification function name: Tmr0CmAIntFunc

☑ Use compare match B interrupt (CMIBn)

Interrupt request destination: CPU

Interrupt notification function name: Tmr0CmBIntFunc

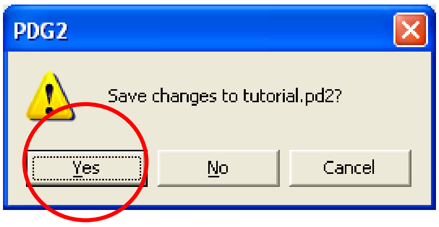
CPU interrupt priority level (Shared with OVIn, CMIAn and CMIBn): 7

-Check compare match A interrupt
Notification function name is
"Tmr0CmAIntFunc"
-Check compare match B interrupt

(8) Generate source files **PDG**

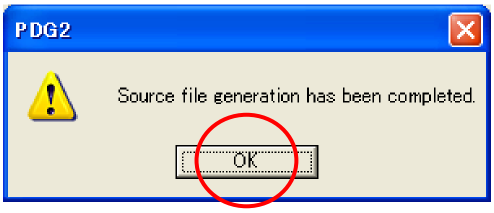
1. To generate source files, click on the tool bar.

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



Save changes to tutorial.pd2?

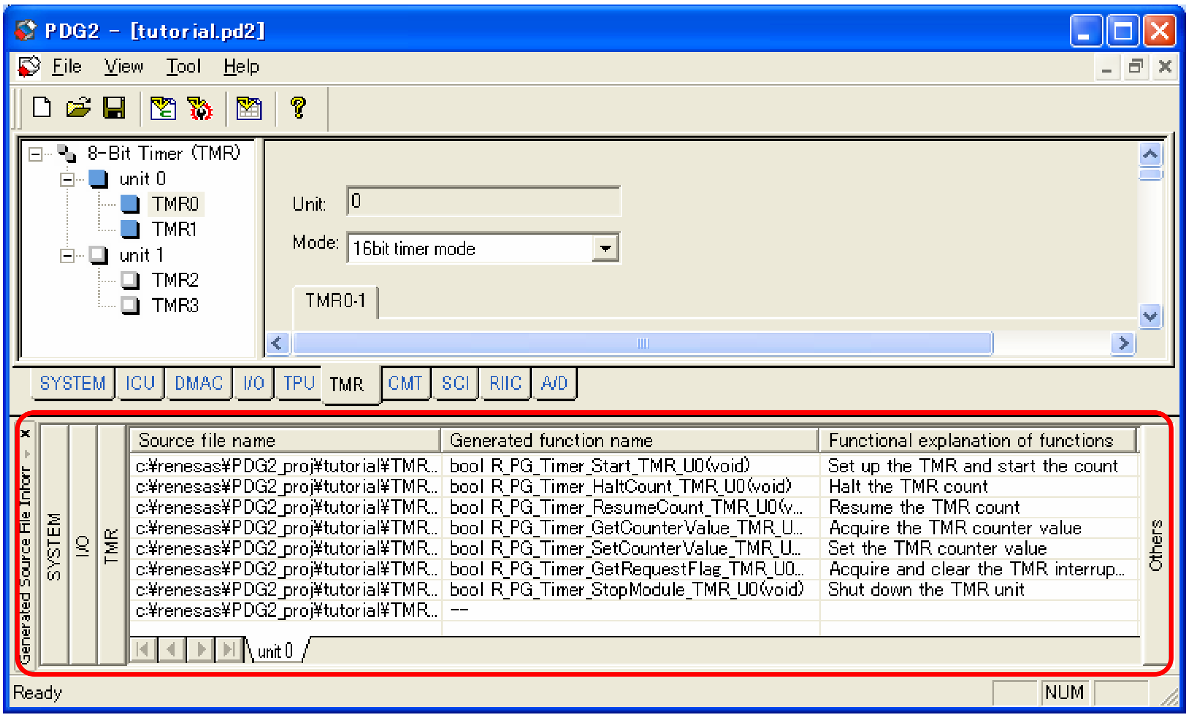
3. Click [OK] on the message box.



Source file generation has been completed.

4. Generated functions are listed in lower pane.

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



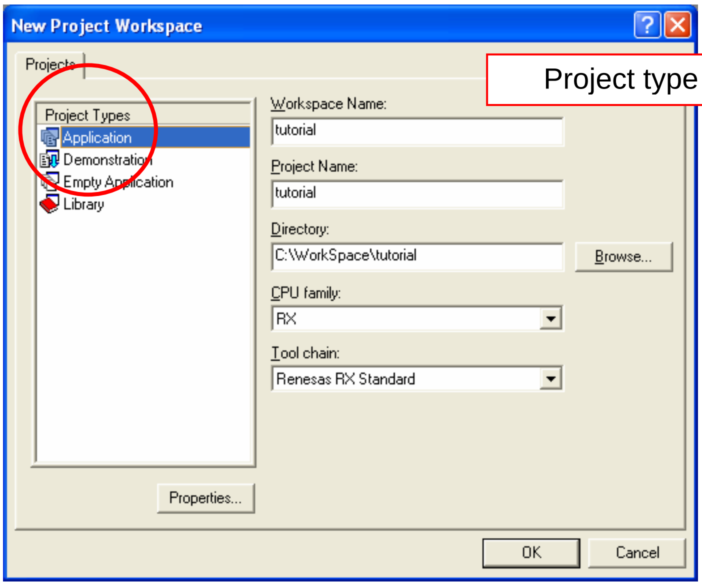
Unit: 0

Mode: 16bit timer mode

Source file name	Generated function name	Functional explanation of functions
c:¥renesas¥PDG2_proj¥tutorial¥TMR...	bool R_PG_Timer_Start_TMR_U0(void)	Set up the TMR and start the count
c:¥renesas¥PDG2_proj¥tutorial¥TMR...	bool R_PG_Timer_HaltCount_TMR_U0(void)	Halt the TMR count
c:¥renesas¥PDG2_proj¥tutorial¥TMR...	bool R_PG_Timer_ResumeCount_TMR_U0(v...	Resume the TMR count
c:¥renesas¥PDG2_proj¥tutorial¥TMR...	bool R_PG_Timer_GetCounterValue_TMR_U...	Acquire the TMR counter value
c:¥renesas¥PDG2_proj¥tutorial¥TMR...	bool R_PG_Timer_SetCounterValue_TMR_U...	Set the TMR counter value
c:¥renesas¥PDG2_proj¥tutorial¥TMR...	bool R_PG_Timer_GetRequestFlag_TMR_U0...	Acquire and clear the TMR interrup...
c:¥renesas¥PDG2_proj¥tutorial¥TMR...	bool R_PG_Timer_StopModule_TMR_U0(void)	Shut down the TMR unit
c:¥renesas¥PDG2_proj¥tutorial¥TMR...	--	

(9) Prepare the HEW project **HEW**

Start the HEW and make RX610 workspace.



New Project Workspace

Projects

Project Types: Application, Demonstration, Empty Application, Library

Workspace Name: tutorial

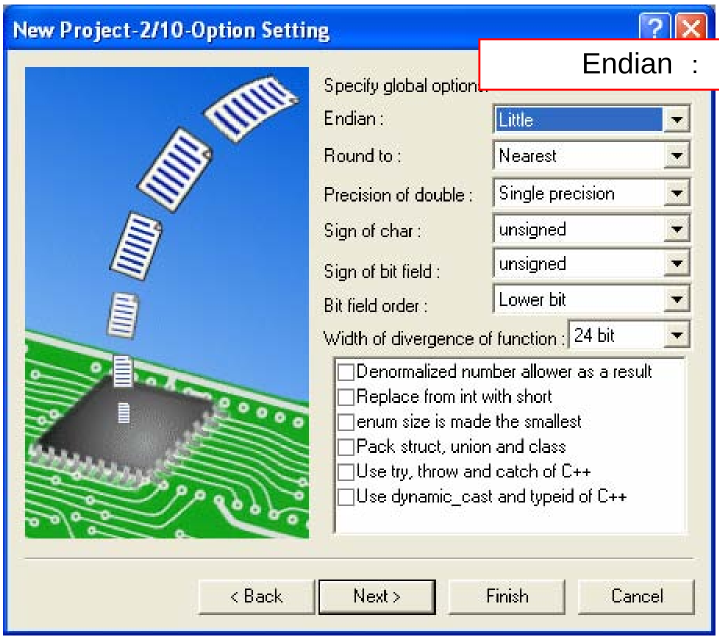
Project Name: tutorial

Directory: C:\WorkSpace\tutorial

CPU family: RX

Tool chain: Renesas RX Standard

Project type : Application



New Project-2/10-Option Setting

Specify global options

Endian : Little

Round to : Nearest

Precision of double : Single precision

Sign of char : unsigned

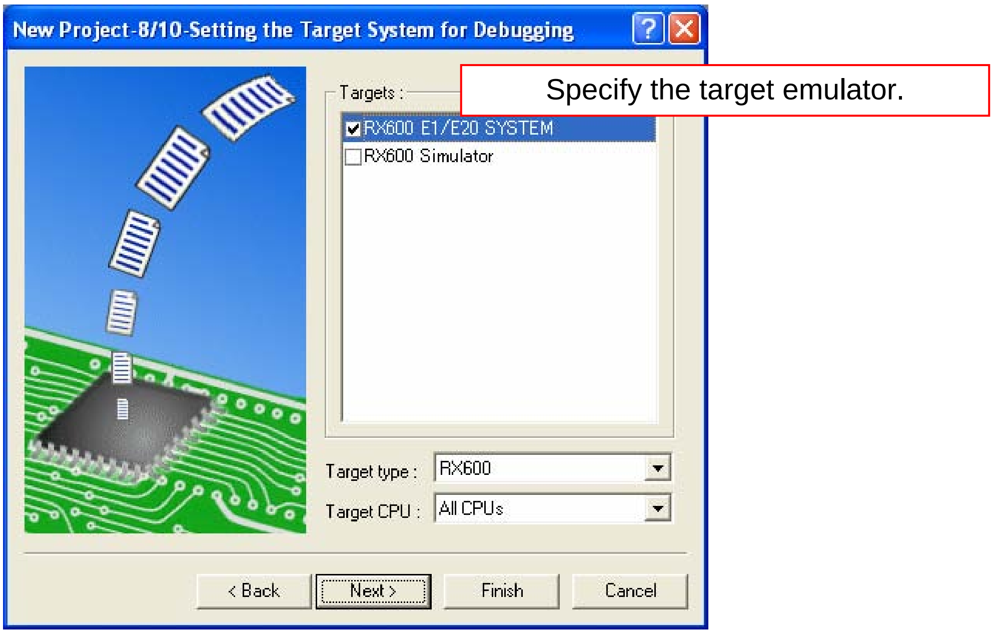
Sign of bit field : unsigned

Bit field order : Lower bit

Width of divergence of function : 24 bit

- ☐ Denormalized number allower as a result
- ☐ Replace from int with short
- ☐ enum size is made the smallest
- ☐ Pack struct, union and class
- ☐ Use try, throw and catch of C++
- ☐ Use dynamic_cast and typeid of C++

Endian : Little



Specify the target emulator.

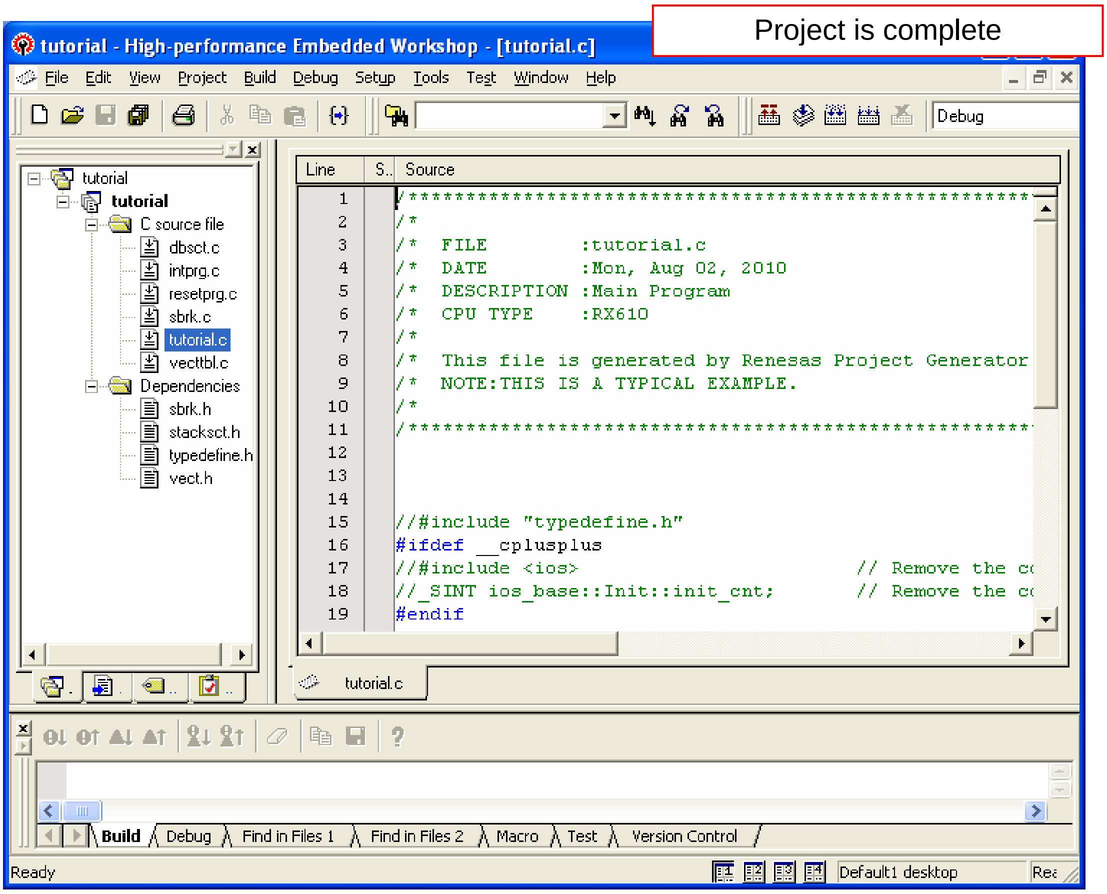
New Project-8/10-Setting the Target System for Debugging

Targets :

☑ RX600 E1/E20 SYSTEM
□ RX600 Simulator

Target type : RX600

Target CPU : All CPUs



Project is complete

tutorial - High-performance Embedded Workshop - [tutorial.c]

```
/*****************************************************************
/*
/*  FILE        :tutorial.c
/*  DATE        :Mon, Aug 02, 2010
/*  DESCRIPTION :Main Program
/*  CPU TYPE    :RX610
/*
/*  This file is generated by Renesas Project Generator
/*  NOTE:THIS IS A TYPICAL EXAMPLE.
/*
/*****************************************************************



//#include "typedefine.h"
#ifdef __cplusplus
//#include <ios>                        // Remove the cc
//_SINT ios_base::Init::init_cnt;       // Remove the cc
#endif
```

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

Make the following program on HEW.

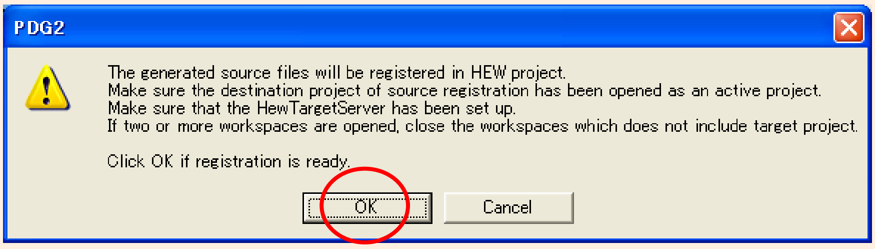
```
//Include "R_PG_<PDG project name>.h"
#include "R_PG_tutorial.h"

void main(void)
{
	//Set Clock
	R_PG_Clock_Set();

	//Set port P33
	R_PG_IO_PORT_Set_P3();

	//Set 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_P33(0);
}

// Compare match B interrupt notification function
void Tmr0CmBIntFunc(void)
{
	// Turn off the LED
	R_PG_IO_PORT_Write_P33(1);
}
```

(11) Add PDG generated source file to HEW

PDG

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

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



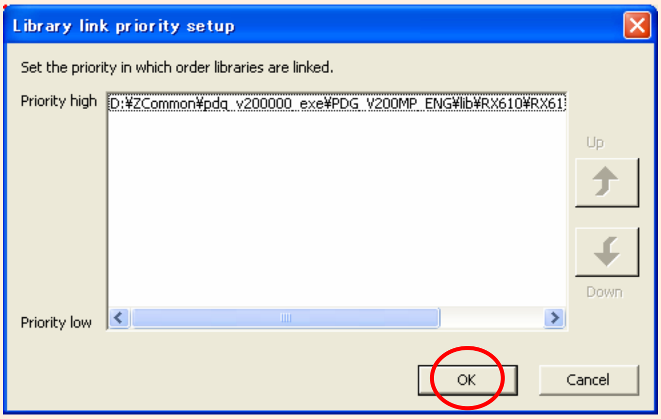
PDG2

The generated source files will be registered in HEW project.
Make sure the destination project of source registration has been opened as an active project.
Make sure that the HewTargetServer has been set up.
If two or more workspaces are opened, close the workspaces which does not include target project.

Click OK if registration is ready.

3. This is a linkage setting of RPDL library.

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



Library link priority setup

Set the priority in which order libraries are linked.

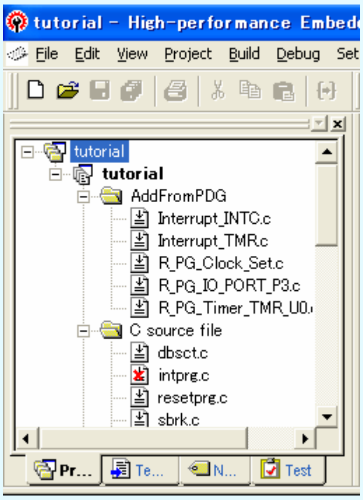
Priority high: D:¥ZCommon¥pdg_v200000_exe¥PDG_V200MP_ENG¥lib¥RX610¥RX61

HEW

4. Source fiels are added to HEW

Added source files are

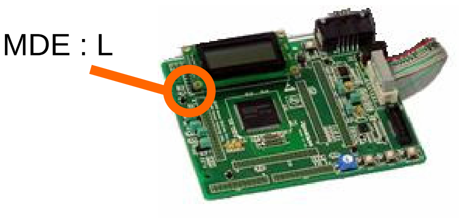
put in "AddFromPDG" folder.



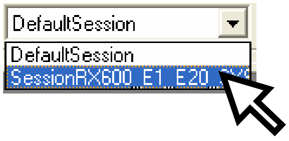
Source files are registered via HEW Target Server.
Make sure that the HEW Target Server has been set up before executing registration.

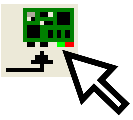
(12) Connect to the emulator, build the program and execute **HEW**

1. Before connecting the emulator, make sure the MDE on RSK board is “L” to set CPU to little endian.



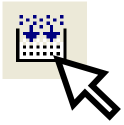
2. Connect to the emulator





Connect button

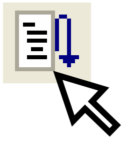
3. Just by clicking [Build] button, program can be built because RPDL library and include directory are automatically registered in build setting.



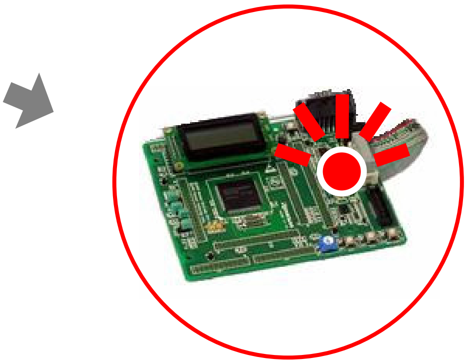
Build button

4. Download the program

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

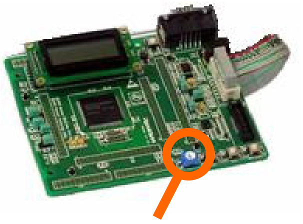


Reset go button



6.2 Execute A/D conversion continuously

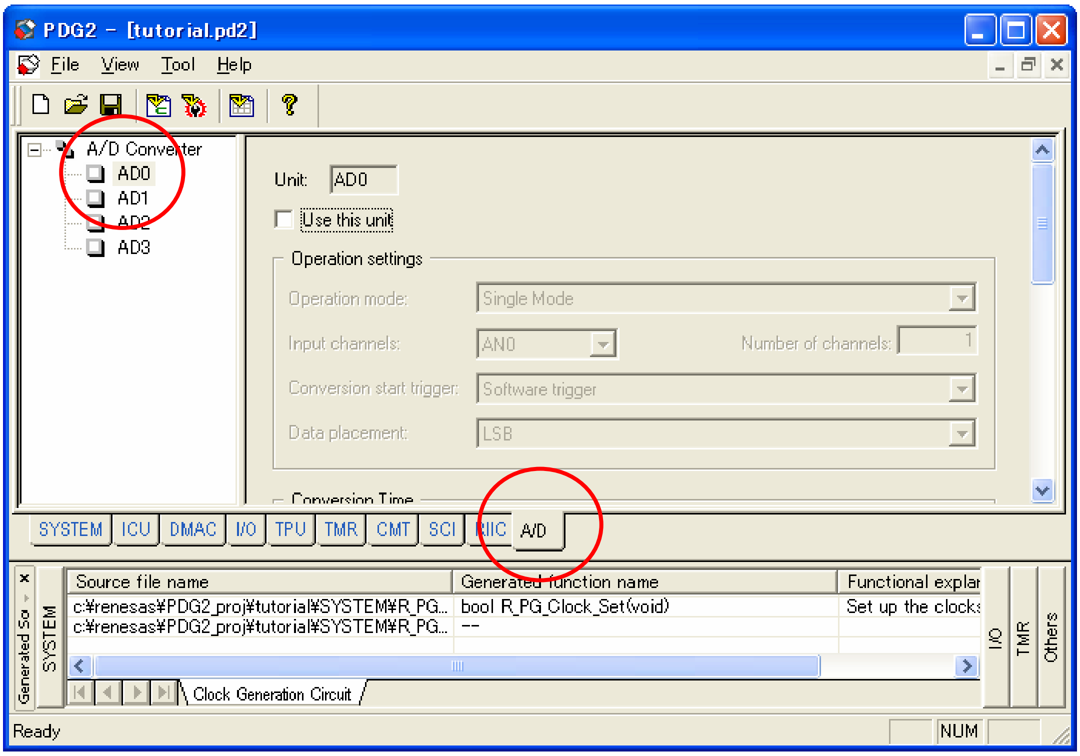
In RX610 RSK board, the potentiometer is connected to AN0 analog input. In this tutorial, set up the AD0 to execute A/D conversion continuously. And check the result of A/D conversion real time on HEW.



(Use the PDG and HEW project made at 6.1, Blink the LED on RSK with TMR interrupt.)

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

Select A/D tab and click AD0 on tree view



Unit: AD0

□ Use this unit

Operation settings

Operation mode: Single Mode

Input channels: AN0 Number of channels: 1

Conversion start trigger: Software trigger

Data placement: LSB

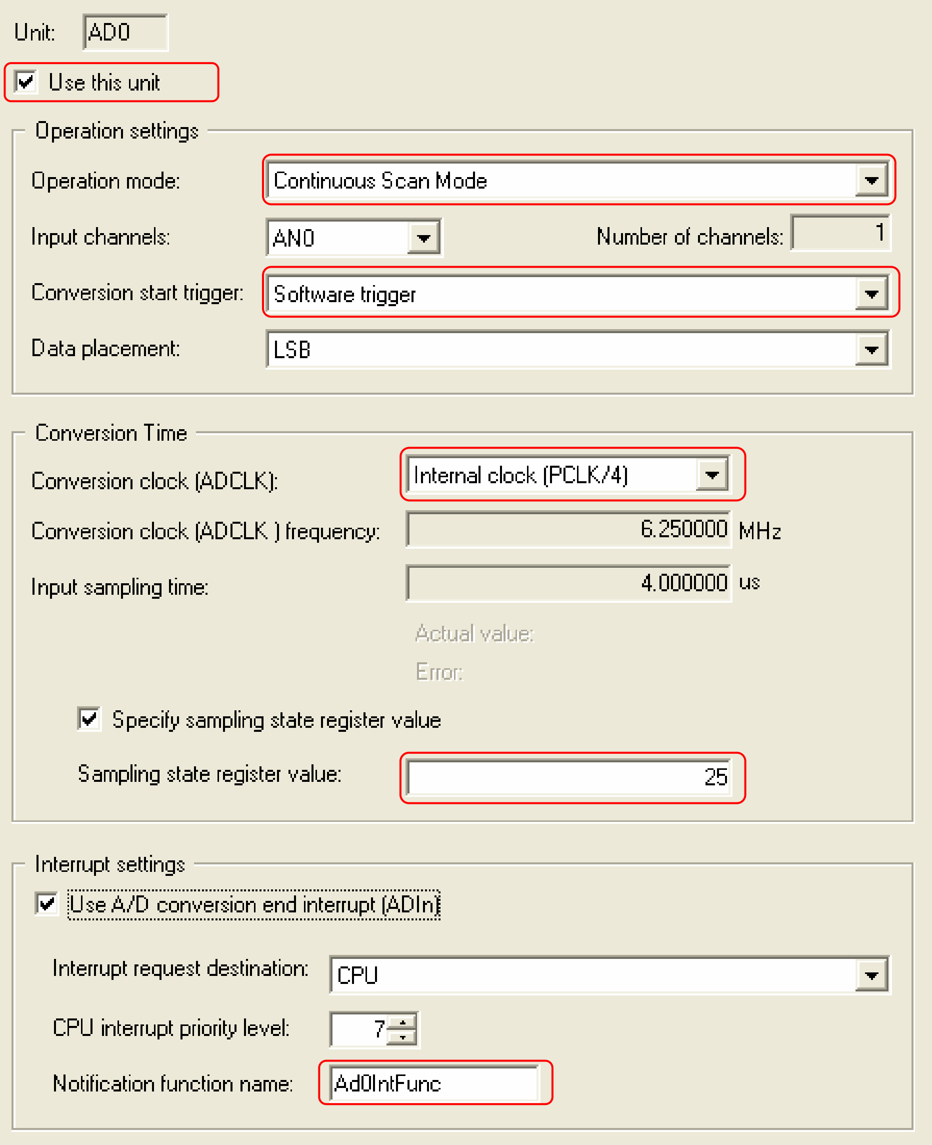
Conversion Time

Source file name	Generated function name	Functional explan
c:¥renesas¥PDG2_proj¥tutorial¥SYSTEM¥R_PG...	bool R_PG_Clock_Set(void)	Set up the clocks
c:¥renesas¥PDG2_proj¥tutorial¥SYSTEM¥R_PG...	--	

(2) A/D converter setting-2 PDG

Make the following setting for AD0.

1. Check "Use this unit"
2. Select "Continuous scan mode"
3. Start trigger is "Software"
4. Use PCLK/4 as conversion clock
5. Leave the default sampling state register value 25.
6. Set A/D conversion end interrupt notification function "Ad0IntFunc".



Unit: AD0

☑ Use this unit

Operation settings

Operation mode: Continuous Scan Mode

Input channels: AN0 Number of channels: 1

Conversion start trigger: Software trigger

Data placement: LSB

Conversion Time

Conversion clock (ADCLK): Internal clock (PCLK/4)

Conversion clock (ADCLK) frequency: 6.250000 MHz

Input sampling time: 4.000000 us

Actual value:

Error:

☑ Specify sampling state register value

Sampling state register value: 25

Interrupt settings

☑ Use A/D conversion end interrupt (ADIn)

Interrupt request destination: CPU

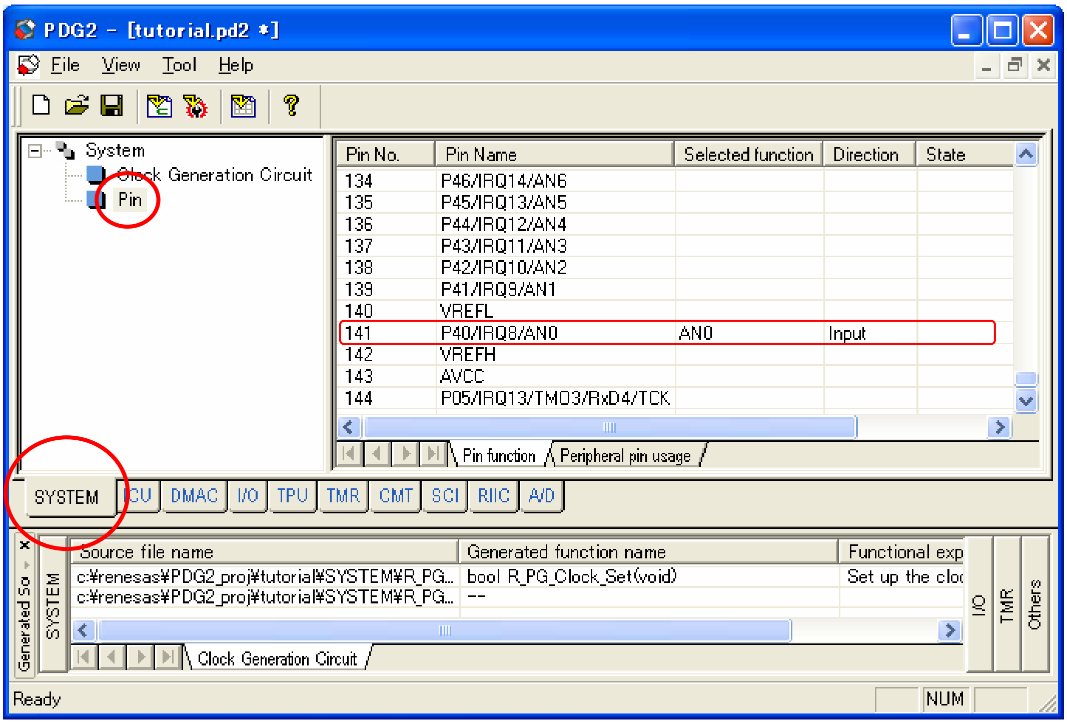
CPU interrupt priority level: 7

Notification function name: Ad0IntFunc

(3) Pin usage check **PDG**

- It is possible to check the usage of pins on the Pin Function Window

1. After setting up the AD0, select SYSTEM tab and click Pin.
2. On the Pin function window, you can see that No.141 pin is used as AN0.

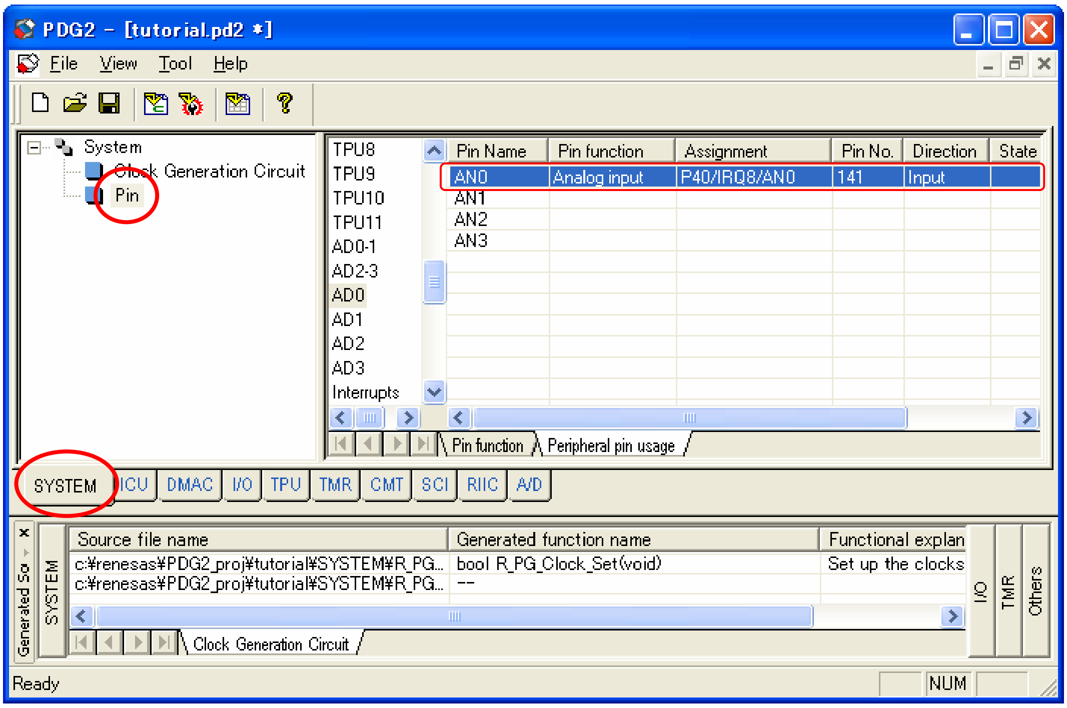


Pin No.	Pin Name	Selected function	Direction	State
134	P46/IRQ14/AN6			
135	P45/IRQ13/AN5			
136	P44/IRQ12/AN4			
137	P43/IRQ11/AN3			
138	P42/IRQ10/AN2			
139	P41/IRQ9/AN1			
140	VREFL			
141	P40/IRQ8/AN0	AN0	Input	
142	VREFH			
143	AVCC			
144	P05/IRQ13/TMO3/RxD4/TCK			

Source file name	Generated function name	Functional exp
c:¥renesas¥PDG2_proj¥tutorial¥SYSTEM¥R_PG...	bool R_PG_Clock_Set(void)	Set up the cloc
c:¥renesas¥PDG2_proj¥tutorial¥SYSTEM¥R_PG...	--	

- 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 pins.



Pin Name	Pin function	Assignment	Pin No.	Direction	State
AN0	Analog input	P40/IRQ8/AN0	141	Input	
AN1					
AN2					
AN3					

Source file name	Generated function name	Functional explan
c:¥renesas¥PDG2_proj¥tutorial¥SYSTEM¥R_PG...	bool R_PG_Clock_Set(void)	Set up the clocks
c:¥renesas¥PDG2_proj¥tutorial¥SYSTEM¥R_PG...	--	

(4) Make the program on HEW

HEW

Make the following program on HEW.

1. Modify the main function as follows.

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

void main(void)
{
	//Set Clock
	R_PG_Clock_Set();

	//Set ADC
	R_PG_ADC_10_Set_AD0();

	//Set port P33
	R_PG_IO_PORT_Set_P3();

	//Set TMR Unit0 and start count
	R_PG_Timer_Start_TMR_U0();

	//Start A/D conversion
	R_PG_ADC_10_StartConversionSW_AD0();

	while(1);
}
```

2. Add the following function.

```
// 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);
}
```

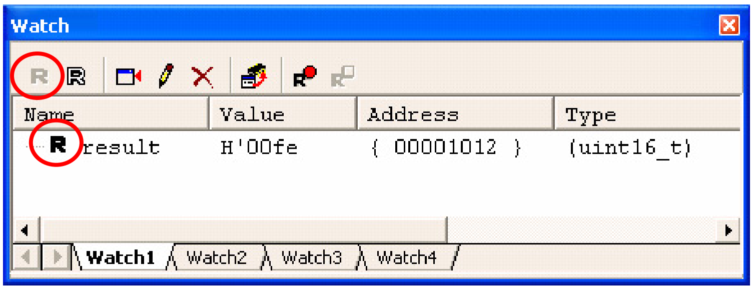
(5) Generate and add the source files to HEW **PDG**

Generate the source fiels and add it to HEW (Refer to 6.1 (8)(11))

(6) Build and execute the program on HEW **HEW**

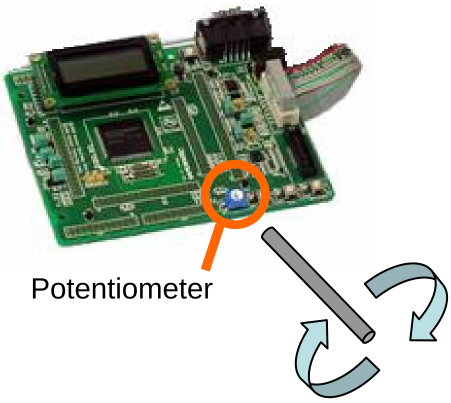
1. Build and download the program
2. Open the Watch window and add the variable "result" to the Watch window

   Set it to the real time update.

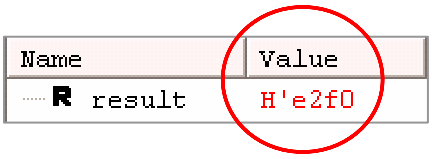


Name	Value	Address	Type
R result	H'00fe	{ 00001012 }	(uint16_t)

3. Eexecute the program
4. Screw the potentiometer to change the analog input voltage during execution.



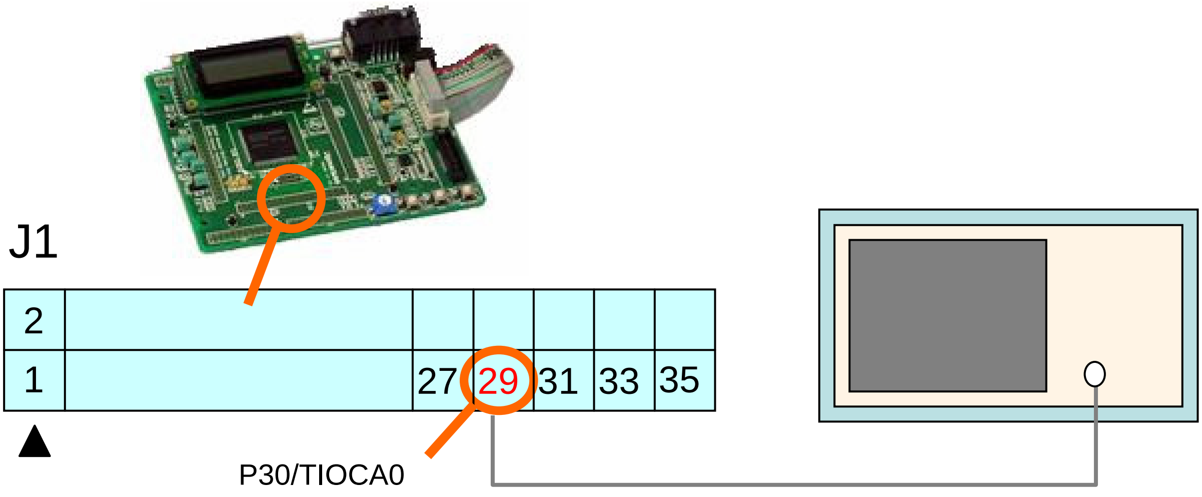
5. The value of “result” on Watch window changes



Name	Value
R result	H'e2f0

6.3 Output PWM pulse with TPU

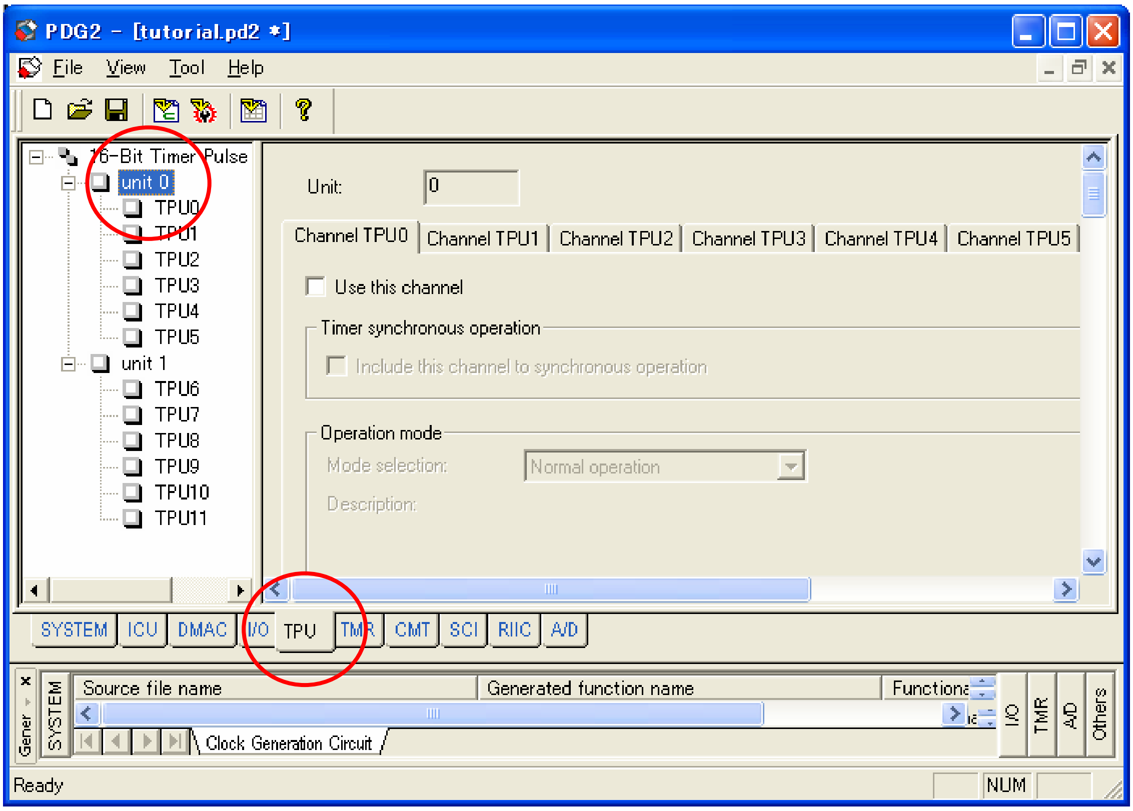
In this tutorial, set up the 16 bit timer pulse unit (TPU) channel 0 to output a pulse to TIOCA0 pin. TIOCA0 is No.29 of J1 connector on RSK board. Connect oscilloscope to TIOCA0 pin.



(Use the PDG and HEW project made until section 6.2.)

(1) TPU setting-1 **PDG**

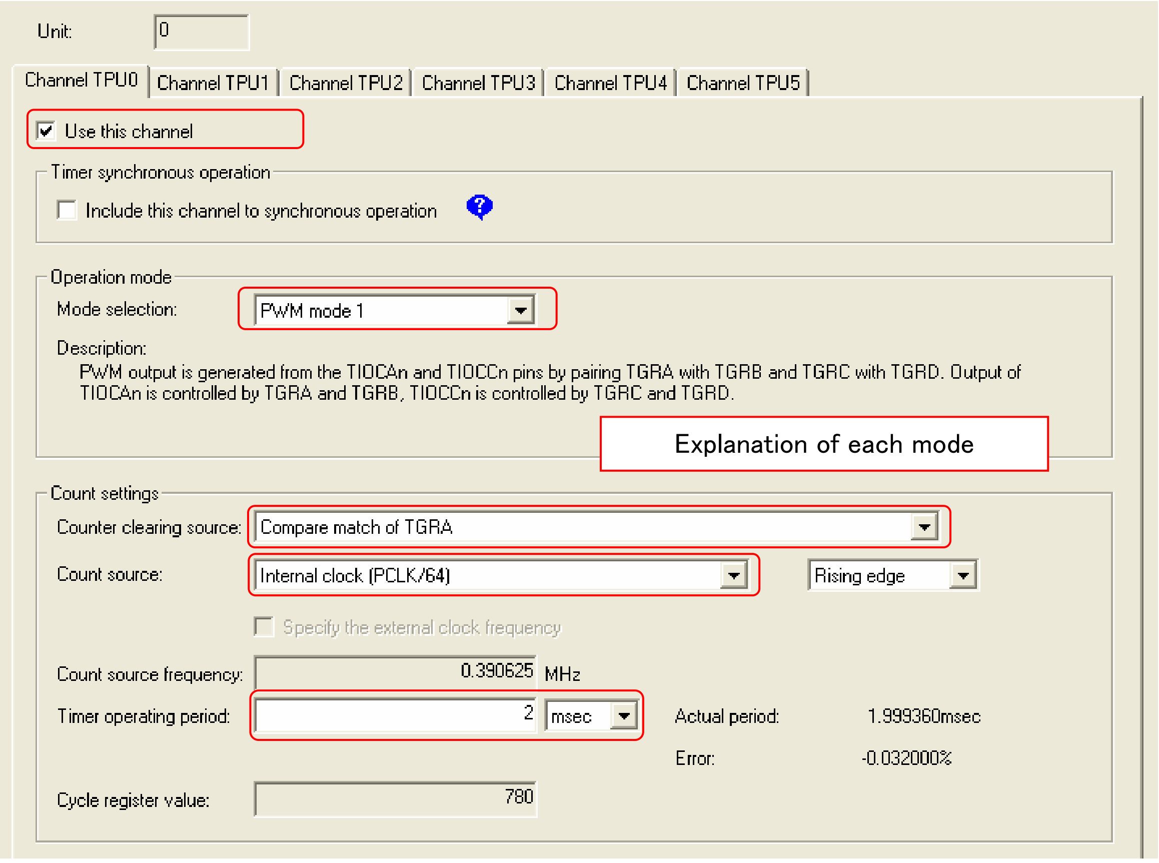
Select TPU tab and click TPU0 on tree view


PDG2 - [tutorial.pd2 *]
File View Tool Help
16-Bit Timer Pulse
unit 0
TPU0
TPU1
TPU2
TPU3
TPU4
TPU5
unit 1
TPU6
TPU7
TPU8
TPU9
TPU10
TPU11
Unit: 0
Channel TPU0 Channel TPU1 Channel TPU2 Channel TPU3 Channel TPU4 Channel TPU5
□ Use this channel
Timer synchronous operation
□ Include this channel to synchronous operation
Operation mode
Mode selection: Normal operation
Description:
SYSTEM ICU DMAC I/O TPU TMR CMT SCI RIIC A/D
Source file name Generated function name Functiona
Clock Generation Circuit
Gener SYSTEM I/O TMR A/D Others
Ready NUM

(2) TPU setting-2 PDG

Make the following setting for TPU0.

1. Check [Use this channel]
2. Select PWM mode 1
3. Select [Compare match A] as a counter clearing source.
4. Select [PCLK/64] as a count source.
5. Set the period to 2 msec



Unit: 0

☑ Use this channel

Timer synchronous operation

□ Include this channel to synchronous operation

Operation mode

Mode selection: PWM mode 1

Description:
PWM output is generated from the TIOCAn and TIOCCn pins by pairing TGRA with TGRB and TGRC with TGRD. Output of TIOCAn is controlled by TGRA and TGRB, TIOCCn is controlled by TGRC and TGRD.

Explanation of each mode

Count settings

Counter clearing source: Compare match of TGRA

Count source: Internal clock (PCLK/64) Rising edge

□ Specify the external clock frequency

Count source frequency: 0.390625 MHz

Timer operating period: 2 msec

Actual period: 1.999360msec

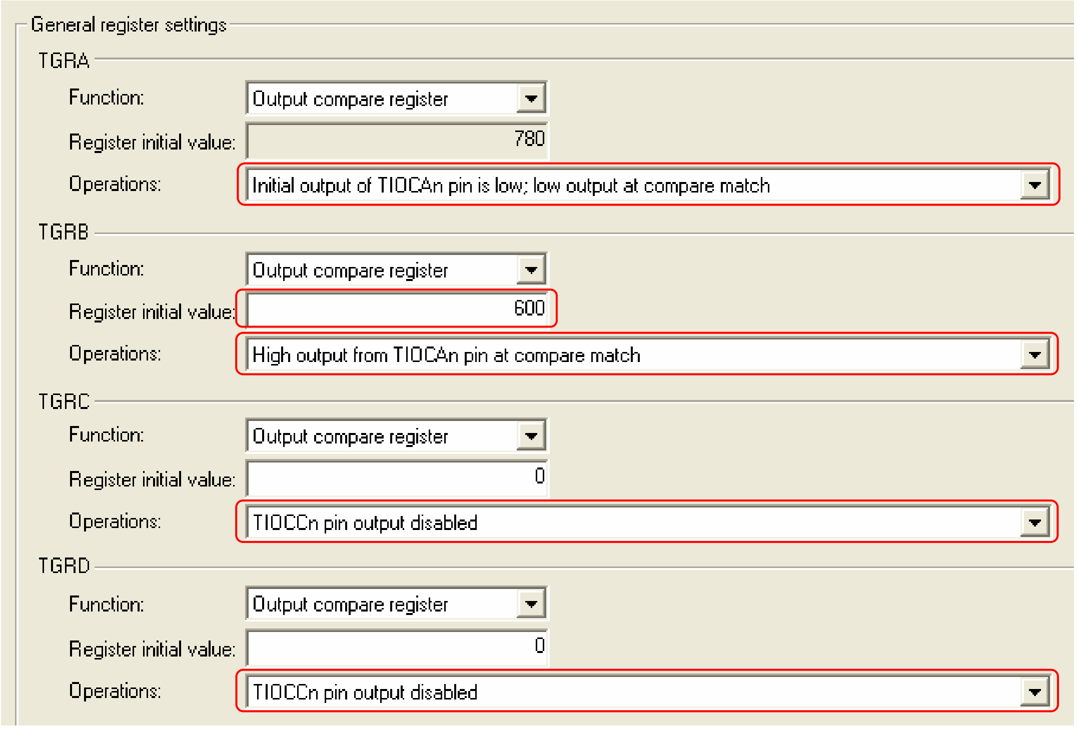
Error: -0.032000%

Cycle register value: 780

(3) TPU setting-3 PDG

Set the output control as follows.

1. Select low output at TGRA compare match
2. Set the TGRB register value (Compare match B value) to 600.
3. Select high output at TGRB compare match
4. Disable the output control of TGRC and TGRD.



General register settings

TGRA
- Function: Output compare register
- Register initial value: 780
- Operations: Initial output of TIOCAn pin is low; low output at compare match

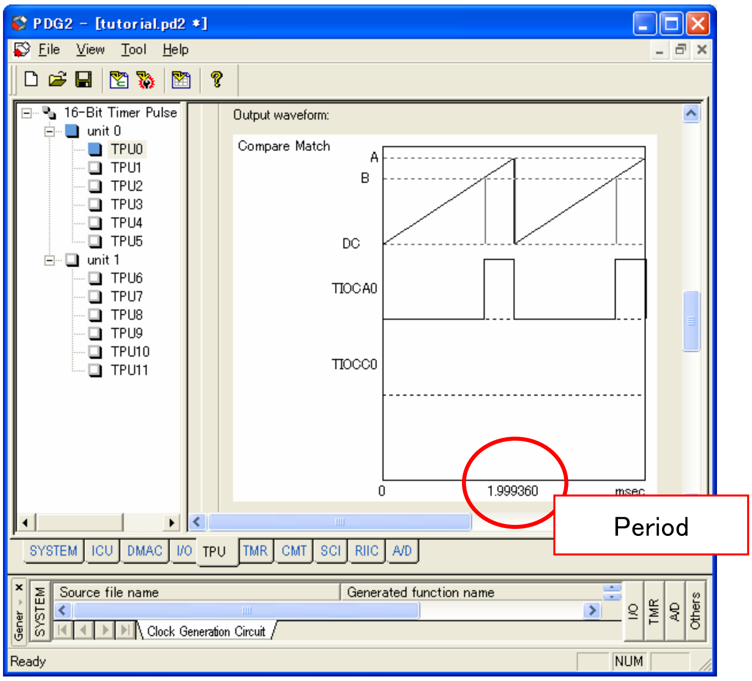
TGRB
- Function: Output compare register
- Register initial value: 600
- Operations: High output from TIOCAn pin at compare match

TGRC
- Function: Output compare register
- Register initial value: 0
- Operations: TIOCCn pin output disabled

TGRD
- Function: Output compare register
- Register initial value: 0
- Operations: TIOCCn pin output disabled

(4) Check the waveform PDG

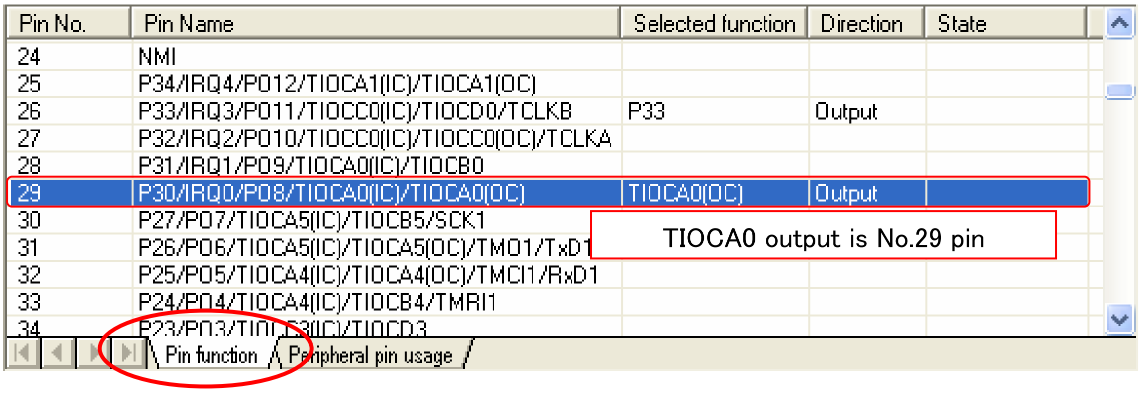
The pulse waveform is displayed



(5) Check the pin usage PDG

Check the status of TIOCA0 pin on the Pin Function Window and the Peripheral Pin Usage Window.

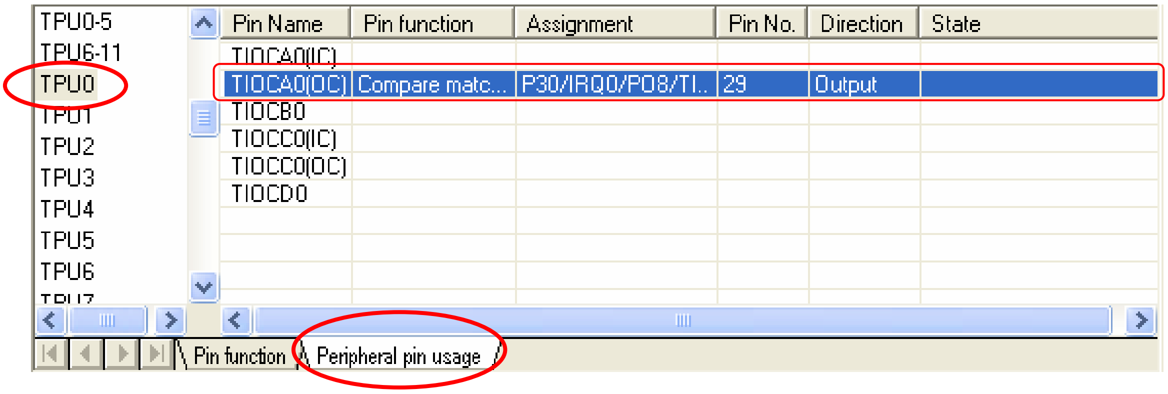
Pin Function Window



Pin No.	Pin Name	Selected function	Direction	State
24	NMI			
25	P34/IRQ4/PO12/TIOCA1(IC)/TIOCA1(OC)			
26	P33/IRQ3/PO11/TIOCC0(IC)/TIOCD0/TCLKB	P33	Output	
27	P32/IRQ2/PO10/TIOCC0(IC)/TIOCC0(OC)/TCLKA			
28	P31/IRQ1/PO9/TIOCA0(IC)/TIOCB0			
29	P30/IRQ0/PO8/TIOCA0(IC)/TIOCA0(OC)	TIOCA0(OC)	Output	
30	P27/PO7/TIOCA5(IC)/TIOCB5/SCK1			
31	P26/PO6/TIOCA5(IC)/TIOCA5(OC)/TMO1/TxD1			
32	P25/PO5/TIOCA4(IC)/TIOCA4(OC)/TMCI1/RxD1			
33	P24/PO4/TIOCA4(IC)/TIOCB4/TMRI1			
34	P23/PO3/TIOCC3(IC)/TIOCD3			

TIOCA0 output is No.29 pin

Peripheral Pin Usage Window



TPU0-5
TPU6-11
TPU0
TPU1
TPU2
TPU3
TPU4
TPU5
TPU6
TPU7

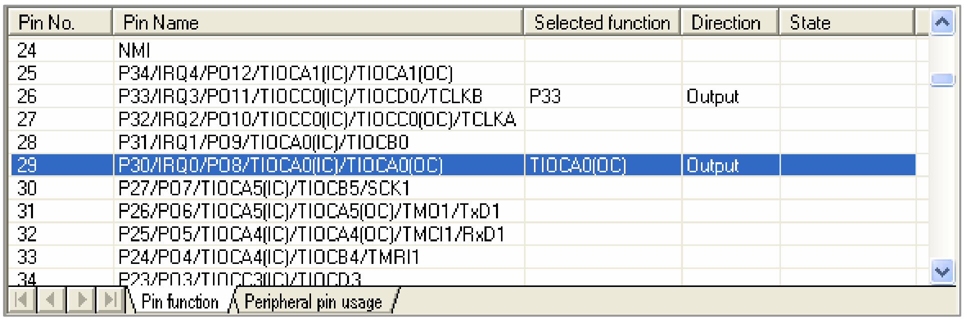
Pin Name	Pin function	Assignment	Pin No.	Direction	State
TIOCA0(IC)					
TIOCA0(OC)	Compare matc...	P30/IRQ0/PO8/TI...	29	Output	
TIOCB0					
TIOCC0(IC)					
TIOCC0(OC)					
TIOCD0					

(6) Output the pin list **PDG**

To output the contents of pin windows to CVS file select [Tool] -> [Generate pin lists] menu or click on the tool bar.

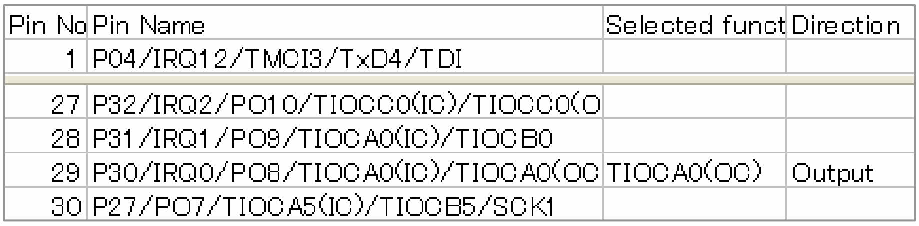
Output directory is "PDG project folder¥PIN".

PinFunctionWindow



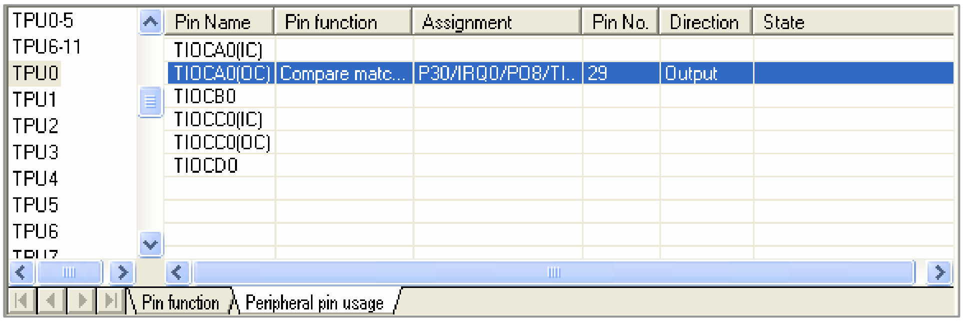
Pin No.	Pin Name	Selected function	Direction	State
24	NMI			
25	P34/IRQ4/P012/TIOCA1(IC)/TIOCA1(OC)			
26	P33/IRQ3/P011/TIOCC0(IC)/TIOCD0/TCLKB	P33	Output	
27	P32/IRQ2/P010/TIOCC0(IC)/TIOCC0(OC)/TCLKA			
28	P31/IRQ1/P09/TIOCA0(IC)/TIOCB0			
29	P30/IRQ0/P08/TIOCA0(IC)/TIOCA0(OC)	TIOCA0(OC)	Output	
30	P27/P07/TIOCA5(IC)/TIOCB5/SCK1			
31	P26/P06/TIOCA5(IC)/TIOCA5(OC)/TM01/TxD1			
32	P25/P05/TIOCA4(IC)/TIOCA4(OC)/TMCI1/RxD1			
33	P24/P04/TIOCA4(IC)/TIOCB4/TMRI1			
34	P23/P03/TIOCC3(IC)/TIOCD3			

PinFUnction.csv



Pin No	Pin Name	Selected funct	Direction
1	P04/IRQ12/TMCI3/TxD4/TDI		
27	P32/IRQ2/PO10/TIOCC0(IC)/TIOCC0(O		
28	P31/IRQ1/PO9/TIOCA0(IC)/TIOCB0		
29	P30/IRQ0/PO8/TIOCA0(IC)/TIOCA0(OC	TIOCA0(OC)	Output
30	P27/PO7/TIOCA5(IC)/TIOCB5/SCK1		

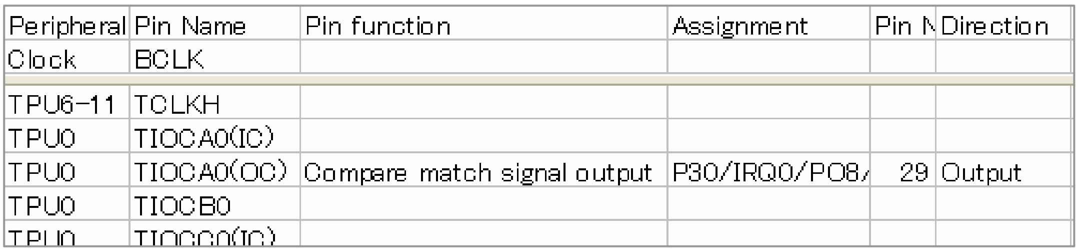
PeripheralPinUsageWindow



TPU0-5
TPU6-11
TPU0
TPU1
TPU2
TPU3
TPU4
TPU5
TPU6
TPU7

Pin Name	Pin function	Assignment	Pin No.	Direction	State
TIOCA0(IC)					
TIOCA0(OC)	Compare matc...	P30/IRQ0/P08/TI...	29	Output	
TIOCB0					
TIOCC0(IC)					
TIOCC0(OC)					
TIOCD0					

PeripheralPinUsage.csv



Peripheral	Pin Name	Pin function	Assignment	Pin N	Direction
Clock	BCLK				
TPU6-11	TCLKH				
TPU0	TIOCA0(IC)				
TPU0	TIOCA0(OC)	Compare match signal output	P30/IRQ0/PO8/	29	Output
TPU0	TIOCB0				
TPU0	TIOCC0(IC)				

(7) Make the program on HEW **HEW**

Make the following program on HEW.

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

void main(void)
{
    //Set Clock
    R_PG_Clock_Set();

    //Set ADC
    R_PG_ADC_10_Set_AD0();

    //Set port P33
    R_PG_IO_PORT_Set_P3();

    //Set TMR Unit0 and start count
    R_PG_Timer_Start_TMR_U0();

    //Start A/D conversion
    R_PG_ADC_10_StartConversionSW_AD0 ();

    //Set up TPU0 and start the count
    R_PG_Timer_Start_TPU_U0_C0();

    while(1);
}
```

(8) Generate and add the source files to HEW **PDG**

Generate the source fiels and add it to HEW (Refer to 6.1 (8)(11))

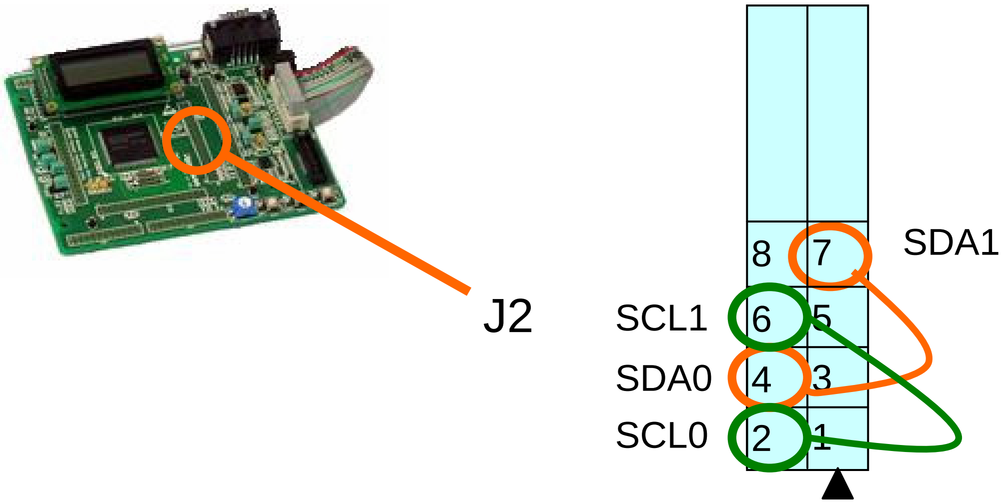
(9) Build and execute the program on HEW **HEW**

1. Build, download and execute the program.
2. Check the output pulse of TIOCA0 by oscilloscope.

6.4 Communicate between I2C channel 0 and channel 1

RX610 has two I2C channels RIIC0 and RIIC1. In this tutorial, set up these channels to transfer data from RIIC0 (master) to RIIC1 (slave).

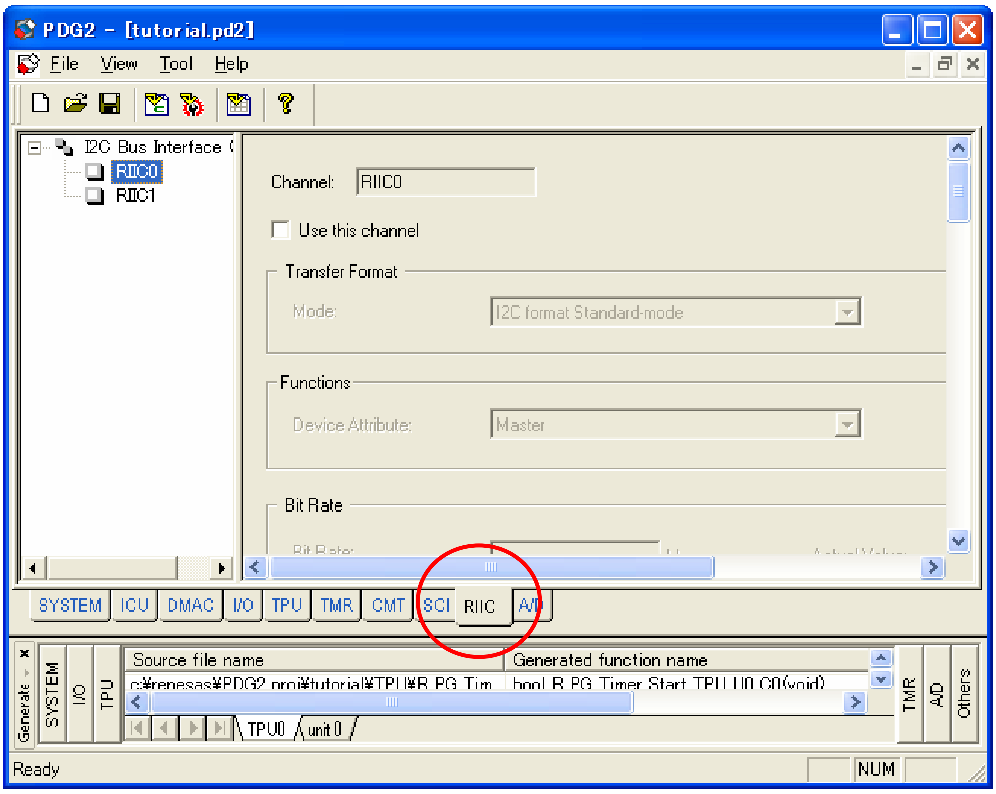
Connect SCL0-SCL1, SDA0-SDA1 on the RSK board. RIICpins are J2/No.2, 4, 6, and 7 on the RSK board.



(Use the PDG and HEW project made until section 6.3.)

(1) RIIC setting **PDG**

Select RIIC tab.


PDG2 - [tutorial.pd2]
File View Tool Help
I2C Bus Interface (
RIIC0
RIIC1
SYSTEM ICU DMAC I/O TPU TMR CMT SCI RIIC A/D
Generate SYSTEM I/O TPU TMR A/D Others
Source file name
Generated function name
c:¥renesas¥PDG2 proj¥tutorial¥TPU¥R PG Tim
bool R PG Timer Start TPU U0 C0(void)
TPU0 unit 0
Ready NUM

Channel: RIIC0

□ Use this channel

Transfer Format

Mode: I2C format Standard-mode

Functions

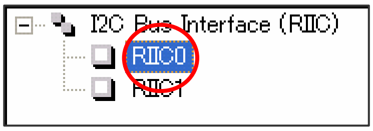
Device Attribute: Master

Bit Rate

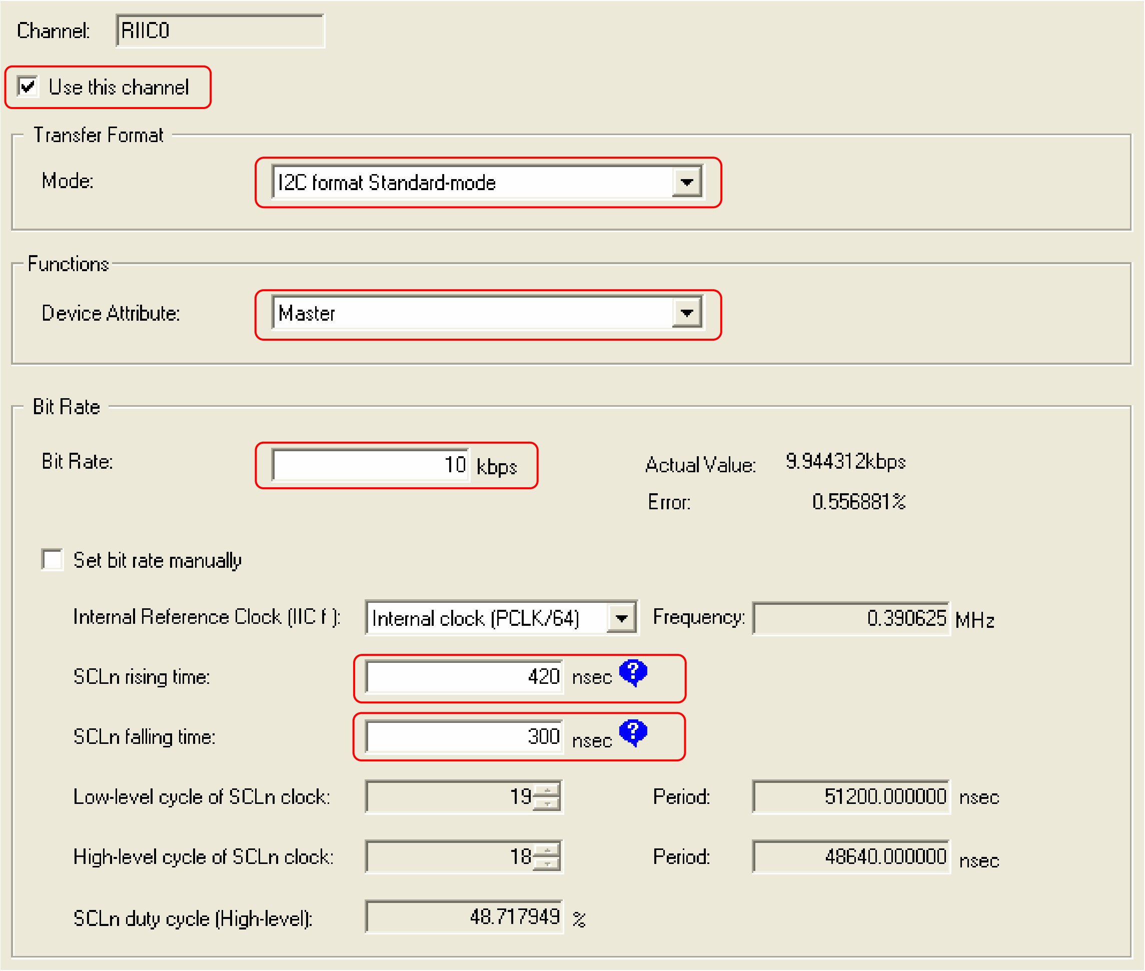
(2) RIIC0 (master) setting **PDG**

Set RIIC0 as follows.

1. Select RIIC0 on the tree view



2. Check [Use this channel]
3. Select [I2C format standard mode]
4. Select [Master] for device attribute
5. Set bit rate to 10 kbps
6. The rise time and fall time of SCLN depend on the HW system. For RSK board, set 420 ns and 300 ns.



Channel: RIIC0

☑ Use this channel

Transfer Format

Mode: I2C format Standard-mode

Functions

Device Attribute: Master

Bit Rate

Bit Rate: 10 kbps

Actual Value: 9.944312kbps

Error: 0.556881%

□ Set bit rate manually

Internal Reference Clock (IIC f): Internal clock (PCLK/64) Frequency: 0.390625 MHz

SCLn rising time: 420 nsec

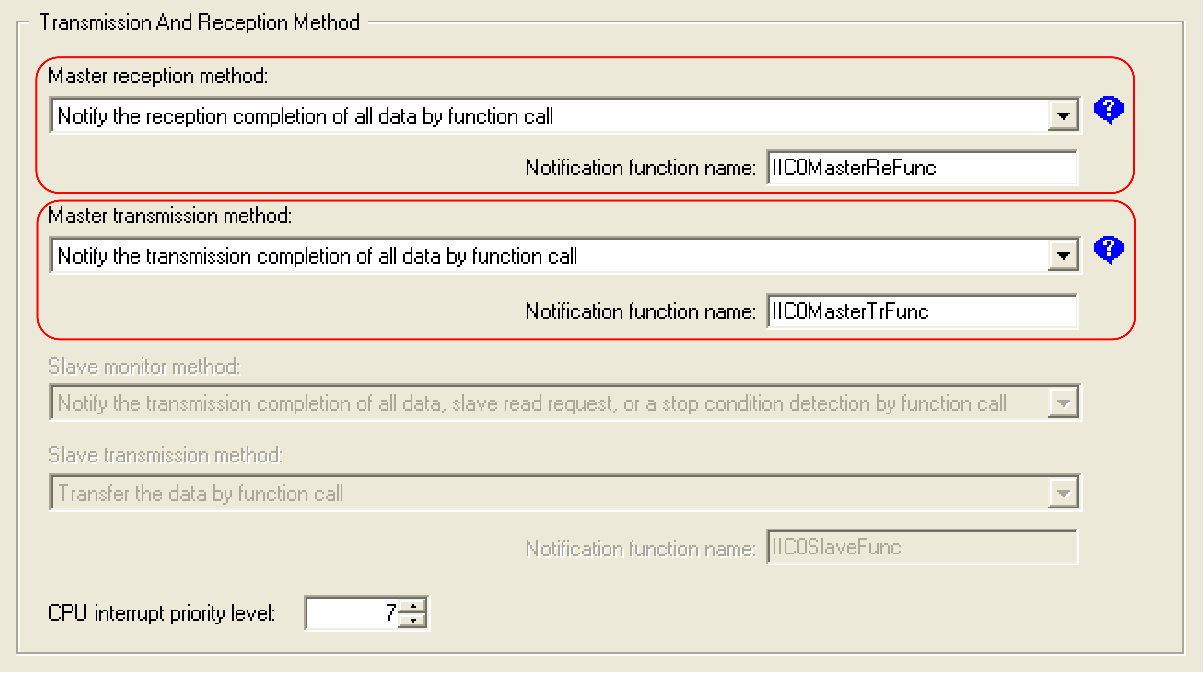
SCLn falling time: 300 nsec

Low-level cycle of SCLn clock: 19 Period: 51200.000000 nsec

High-level cycle of SCLn clock: 18 Period: 48640.000000 nsec

SCLn duty cycle (High-level): 48.717949 %

7. Select [Notify the reception completion of all data by function call] as the master reception method. Specify “IIC0MasterReFunc” as a notification function name.
8. Select [Notify the transmission completion of all data by function call] as the master transmission method. Specify “IIC0MasterTrFunc” as a notification function name.



Transmission And Reception Method

Master reception method: Notify the reception completion of all data by function call

Notification function name: IIC0MasterReFunc

Master transmission method: Notify the transmission completion of all data by function call

Notification function name: IIC0MasterTrFunc

Slave monitor method: Notify the transmission completion of all data, slave read request, or a stop condition detection by function call

Slave transmission method: Transfer the data by function call

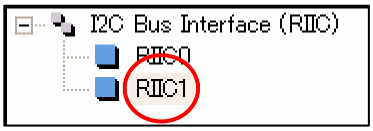
Notification function name: IIC0SlaveFunc

CPU interrupt priority level: 7

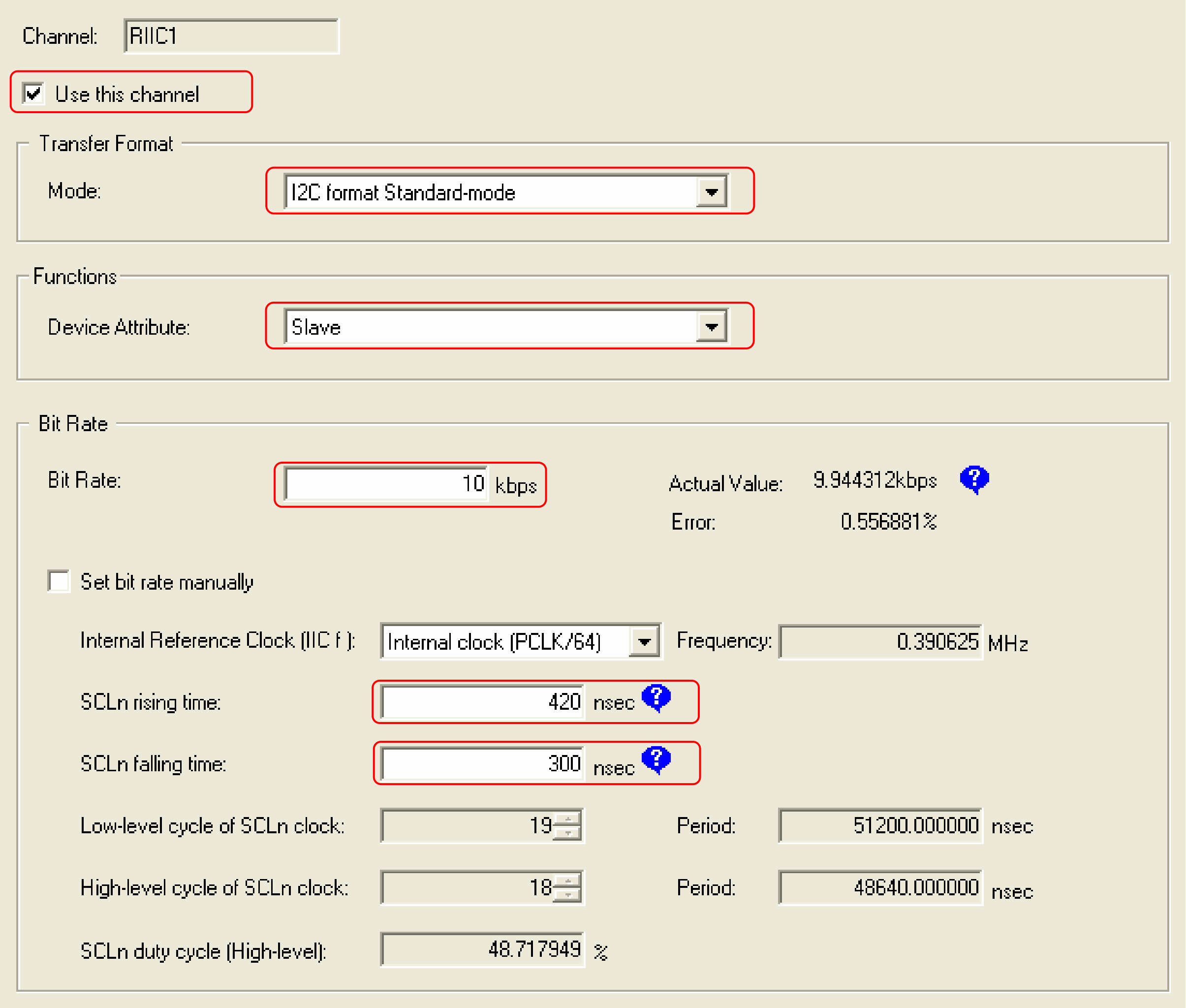
(3) RIIC1 (Slave) setting **PDG**

Set RIIC1 as follows.

1. Select RIIC1 on the tree view



2. Check [Use this channel]
3. Select [I2C format standard mode]
4. Select [Slave] for device attribute
5. Set bit rate is same asRIIC0
6. SCLn rise time and fall time are same as RIIC0.



Channel: RIIC1

☑ Use this channel

Transfer Format

Mode: I2C format Standard-mode

Functions

Device Attribute: Slave

Bit Rate

Bit Rate: 10 kbps

Actual Value: 9.944312kbps

Error: 0.556881%

□ Set bit rate manually

Internal Reference Clock (IIC f): Internal clock (PCLK/64)

Frequency: 0.390625 MHz

SCLn rising time: 420 nsec

SCLn falling time: 300 nsec

Low-level cycle of SCLn clock: 19

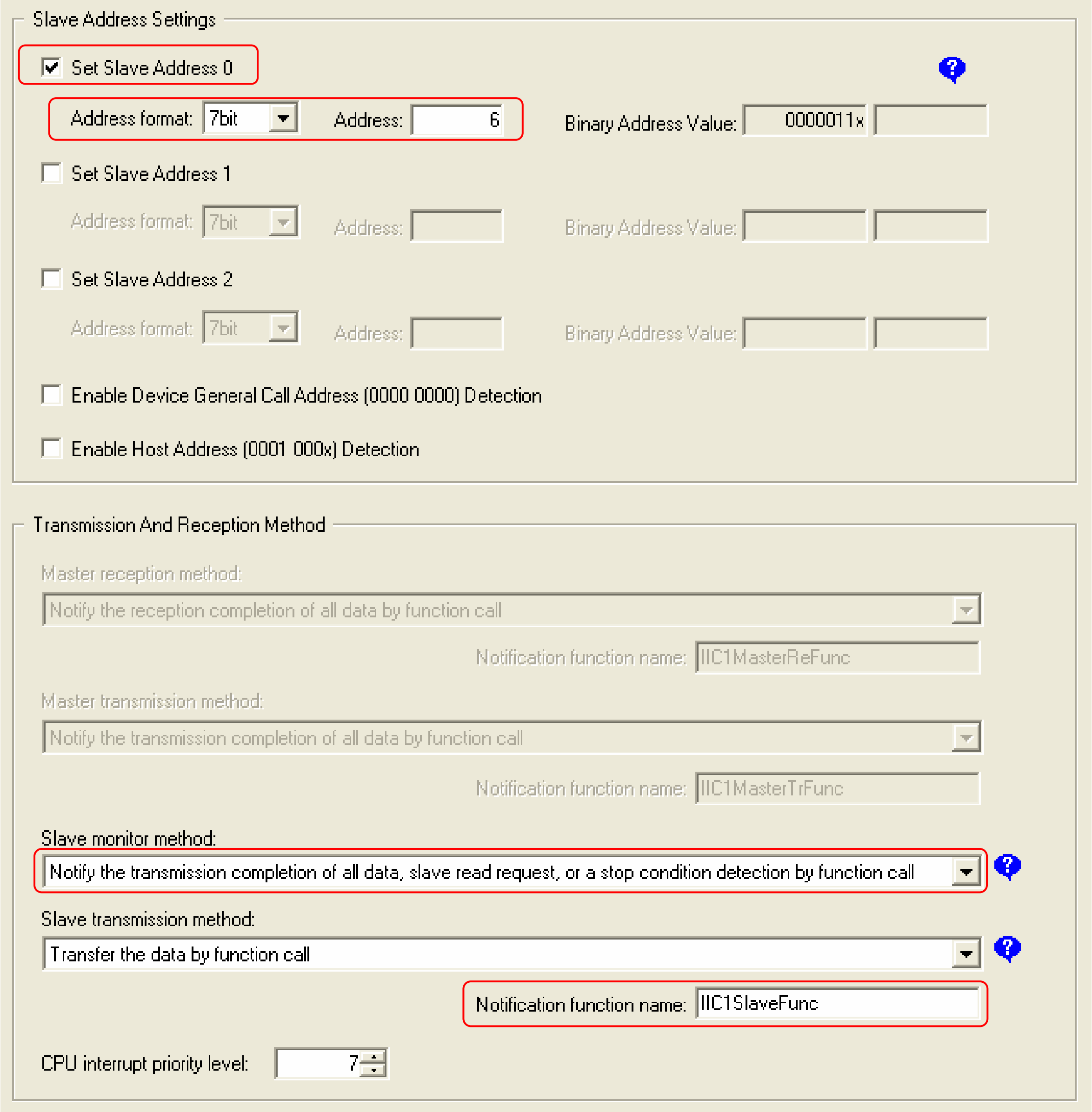
Period: 51200.000000 nsec

High-level cycle of SCLn clock: 18

Period: 48640.000000 nsec

SCLn duty cycle (High-level): 48.717949 %

7. Set the slave address to 0 (7 bit)
8. Select [Notify the transmission completion of all data, slave read request, or a stop condition detection by function call] as the slave monitor method. Specify “IIC1SlaveFunc” as a notification function name.



Slave Address Settings

☑ Set Slave Address 0

Address format: 7bit Address: 6 Binary Address Value: 0000011x

□ Set Slave Address 1

Address format: 7bit Address: Binary Address Value:

□ Set Slave Address 2

Address format: 7bit Address: Binary Address Value:

□ Enable Device General Call Address (0000 0000) Detection

□ Enable Host Address (0001 000x) Detection

Transmission And Reception Method

Master reception method: Notify the reception completion of all data by function call

Notification function name: IIC1MasterReFunc

Master transmission method: Notify the transmission completion of all data by function call

Notification function name: IIC1MasterTrFunc

Slave monitor method: Notify the transmission completion of all data, slave read request, or a stop condition detection by function call

Slave transmission method: Transfer the data by function call

Notification function name: IIC1SlaveFunc

CPU interrupt priority level: 7

(4) Make the program on HEW **HEW**

Make the following program on HEW.

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

uint8_t tr[]="renesas";
uint8_t re[]="----------";

void main(void)
{
    //Set Clock
    R_PG_Clock_Set();

    //Set RIIC0 ans RIIC1
    R_PG_I2C_Set_C0();
    R_PG_I2C_Set_C1();

    //RIIC0 Slave Monitor (Wait receiving)
    R_PG_I2C_SlaveMonitor_C1(
        re, //Storage area of data
        8 //Number of data to be receive
    );

    //RIIC0 Master Send
    R_PG_I2C_MasterSend_C0(
        6, //Slave address
        tr, //Start address of the data to be sent
        8 //Number of the data to be sent
    );
    while(1);
}
```

```
uint16_t tr_count;
uint16_t re_count;

//Master transmission notification function
void IIC0MasterTrFunc(void)
{
    R_PG_I2C_GetSentDataCount_C0(&tr_count);
}

//Master reception notification function
void IIC0MasterReFunc(void)
{
}

//Slave monitor notification function
void IIC1SlaveFunc (void)
{
    R_PG_I2C_GetReceivedDataCount_C1(& re_count);
}
```

(5) Generate and add the source files to HEW **PDG**

Generate the source fiels and add it to HEW (Refer to 6.1 (8)(11))

(6) Build and execute the program on HEW **HEW**

1. Build, download and execute the program.
2. Check the value of reception data “re” on watch window.

RX610 Group
Peripheral Driver Generator
Reference Manual

Publication Date: Feb 16, 2011 Rev.1.01

Published by: Renesas Electronics Corporation

Edited by: Microcomputer Tool Development Department 1
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.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
7F, No. 363 Fu Shing North Road Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 1.0

RX610 Group
Peripheral Driver Generator
Reference Manual

Renesas Electronics Corporation