

# Renesas Peripheral Driver Library

## User's Manual

### RX62G, RX62T Group

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 Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website (<http://www.renesas.com>).

## Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.  
"Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.  
"High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.  
  
Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

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

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

## General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this manual, refer to the relevant sections of the manual. If the descriptions under General Precautions in the Handling of MPU/MCU Products and in the body of the manual differ from each other, the description in the body of the manual takes precedence.

### 1. Handling of Unused Pins

Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.

- The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

### 2. Processing at Power-on

The state of the product is undefined at the moment when power is supplied.

- The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.

In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.

In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

### 3. Prohibition of Access to Reserved Addresses

Access to reserved addresses is prohibited.

- The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

### 4. Clock Signals

After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

- When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

### 5. Differences between Products

Before changing from one product to another, i.e. to one with a different part number, confirm that the change will not lead to problems.

- The characteristics of MPU/MCU in the same group but having different part numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different part numbers, implement a system-evaluation test for each of the products.

## Table of Contents

1. Introduction .....	1
1.1. Tool chain requirements .....	2
1.2. Using the library within your project .....	2
1.2.1. Via the PDG graphical utility .....	2
1.2.2. Added to a project by the user and used stand-alone .....	2
1) Unzip the RPDL files .....	2
2) Copy the files into your project area .....	2
3) Include the new directory .....	5
4) Add the RPDL library file .....	6
5) Include the new source files .....	7
6) Peripherals that are not required .....	8
7) Peripherals that are not supported by RPDL .....	8
8) Avoid conflicts with standard project files .....	9
9) Set the build options .....	11
10) Build the project .....	13
1.2.3. Header file inclusion .....	14
1.2.4. Header file order .....	14
1.3. Document structure .....	15
1.4. List of Abbreviations and Acronyms .....	16
2. Driver .....	17
2.1. Overview .....	17
2.2. Control Functions summary .....	17
2.3. Clock Generation Circuit Driver .....	19
2.4. Interrupt Control Driver .....	20
2.5. I/O Port Driver .....	21
2.6. Port Function Control Driver .....	22
2.7. MCU Operation Driver .....	23
2.8. Low Power Consumption Driver .....	24
2.9. Voltage Detection Circuit Driver .....	25
2.10. Bus Controller Driver .....	26
2.11. Data Transfer Controller Driver .....	27
2.12. Multi-Function Timer Pulse Unit Driver .....	28
2.13. Port Output Enable Driver .....	29
2.14. General PWM Timer Driver .....	30
2.15. Compare Match Timer Driver .....	31
2.16. Watchdog Timer Driver .....	32
2.17. Independent Watchdog Timer Driver .....	33
2.18. Serial Communication Interface Driver .....	34
2.19. CRC Calculator Driver .....	35
2.20. I <sup>2</sup> C Bus Interface Driver .....	36
2.21. Serial Peripheral Interface Driver .....	37
2.22. LIN module .....	38

2.23.	12-bit Analog to Digital Converter Driver .....	39
2.24.	10-bit Analog to Digital Converter Driver .....	40
3.	Types and definitions .....	41
3.1.	Data types.....	41
3.2.	General definitions.....	41
3.2.1.	PDL_NO_FUNC.....	41
3.2.2.	PDL_NO_PTR .....	41
3.2.3.	PDL_NO_DATA.....	41
3.2.4.	PDL_MCU_GROUP.....	41
3.2.5.	PDL_VERSION.....	41
3.2.6.	Bit definitions.....	41
4.	Library Reference.....	42
4.1.	API List by Peripheral Function .....	42
4.2.	Description of Each API.....	44
4.2.1.	Clock Generation Circuit .....	45
1)	R_CGC_Set .....	45
2)	R_CGC_GetStatus .....	46
4.2.2.	Interrupt Control Unit.....	47
1)	R_INTC_CreateExtInterrupt .....	47
2)	R_INTC_CreateSoftwareInterrupt .....	49
3)	R_INTC_CreateFastInterrupt.....	50
4)	R_INTC_CreateExceptionHandlers.....	53
5)	R_INTC_ControlExtInterrupt.....	54
6)	R_INTC_GetExtInterruptStatus .....	56
7)	R_INTC_Read .....	60
8)	R_INTC_Write.....	61
9)	R_INTC_Modify .....	62
4.2.3.	I/O Port.....	63
1)	R_IO_PORT_Set .....	64
2)	R_IO_PORT_ReadControl .....	65
3)	R_IO_PORT_ModifyControl .....	66
4)	R_IO_PORT_Read .....	68
5)	R_IO_PORT_Write .....	69
6)	R_IO_PORT_Compare.....	70
7)	R_IO_PORT_Modify .....	71
8)	R_IO_PORT_Wait .....	72
4.2.4.	Port Function Control.....	73
1)	R_PFC_Read.....	74
2)	R_PFC_Write.....	75
3)	R_PFC_Modify.....	76
4.2.5.	MCU operation .....	77
1)	R_MCU_Control.....	77
2)	R_MCU_GetStatus .....	78
4.2.6.	Low Power Consumption .....	79
1)	R_LPC_Create.....	79
2)	R_LPC_Control.....	81
3)	R_LPC_WriteBackup .....	82
4)	R_LPC_ReadBackup.....	83
5)	R_LPC_GetStatus.....	84
4.2.7.	Voltage Detection Circuit.....	85
1)	R_LVD_Control .....	85
4.2.8.	Bus Controller .....	86
1)	R_BSC_Create .....	86
2)	R_BSC_Control .....	87
3)	R_BSC_GetStatus .....	88

4.2.9.	Data Transfer Controller.....	89
1)	R_DTC_Set.....	89
2)	R_DTC_Create .....	90
3)	R_DTC_Destroy.....	94
4)	R_DTC_Control .....	95
5)	R_DTC_GetStatus .....	97
4.2.10.	Multi-Function Timer Pulse Unit.....	99
1)	R_MTU3_Set .....	99
2)	R_MTU3_Create.....	100
3)	R_MTU3_Destroy .....	110
4)	R_MTU3_ControlChannel .....	111
5)	R_MTU3_ControlUnit .....	114
6)	R_MTU3_ReadChannel .....	121
7)	R_MTU3_ReadUnit .....	123
4.2.11.	Port Output Enable .....	124
1)	R_POE_Set .....	124
2)	R_POE_Create .....	128
3)	R_POE_Control .....	130
4)	R_POE_GetStatus .....	132
4.2.12.	General PWM Timer .....	133
1)	R_GPT_Set.....	133
2)	R_GPT_Create .....	135
3)	R_GPT_Destroy.....	143
4)	R_GPT_ControlChannel.....	144
5)	R_GPT_ControlUnit .....	148
6)	R_GPT_ReadChannel.....	150
7)	R_GPT_ReadUnit.....	153
8)	R_GPT_EdgeDelay_Create .....	155
9)	R_GPT_EdgeDelay_Control.....	157
10)	R_GPT_EdgeDelay_Destroy.....	159
4.2.13.	Compare Match Timer .....	160
1)	R_CMT_Create.....	160
2)	R_CMT_CreateOneShot .....	162
3)	R_CMT_Destroy .....	164
4)	R_CMT_Control .....	165
5)	R_CMT_Read.....	166
4.2.14.	Watchdog Timer .....	167
1)	R_WDT_Create .....	167
2)	R_WDT_Control.....	169
3)	R_WDT_Read.....	170
4.2.15.	Independent Watchdog Timer.....	171
1)	R_IWDT_Set.....	171
2)	R_IWDT_Control.....	172
3)	R_IWDT_Read.....	173
4.2.16.	Serial Communication Interface.....	174
1)	R_SCI_Set .....	174
2)	R_SCI_Create.....	175
3)	R_SCI_Destroy .....	178
4)	R_SCI_Send .....	179
5)	R_SCI_Receive .....	181
6)	R_SCI_Control.....	183
7)	R_SCI_GetStatus.....	185
4.2.17.	CRC calculator.....	187
1)	R_CRC_Create.....	187
2)	R_CRC_Destroy .....	188
3)	R_CRC_Write .....	189
4)	R_CRC_Read.....	190
4.2.18.	I <sup>2</sup> C Bus Interface .....	191
1)	R_IIC_Create .....	191
2)	R_IIC_Destroy .....	195

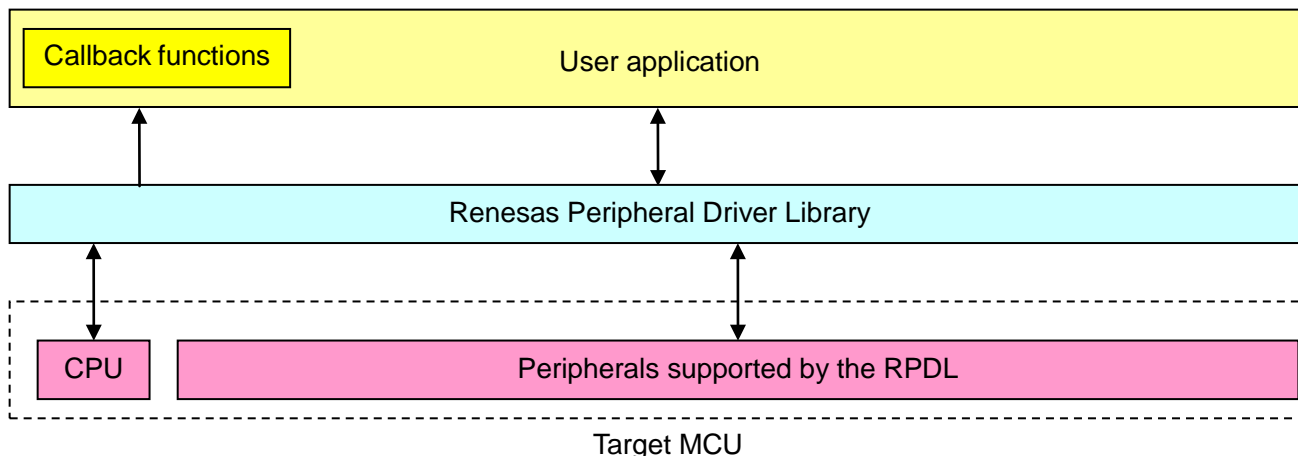
3)	R_IIC_MasterSend .....	196
4)	R_IIC_MasterReceive .....	198
5)	R_IIC_MasterReceiveLast .....	200
6)	R_IIC_SlaveMonitor .....	201
7)	R_IIC_SlaveSend .....	203
8)	R_IIC_Control .....	204
9)	R_IIC_GetStatus .....	205
4.2.19.	Serial Peripheral Interface .....	207
1)	R_SPI_Create .....	207
2)	R_SPI_Destroy .....	210
3)	R_SPI_Command .....	211
4)	R_SPI_Transfer .....	213
5)	R_SPI_Control .....	215
6)	R_SPI_GetStatus .....	217
4.2.20.	LIN module .....	218
1)	R_LIN_Create .....	218
2)	R_LIN_Destroy .....	221
3)	R_LIN_Transfer .....	222
4)	R_LIN_Read .....	224
5)	R_LIN_Control .....	225
6)	R_LIN_GetStatus .....	227
4.2.21.	12-bit Analog to Digital Converter .....	228
1)	R_ADC_12_Set .....	228
2)	R_ADC_12_CreateUnit .....	229
3)	R_ADC_12_CreateChannel .....	234
4)	R_ADC_12_Destroy .....	237
5)	R_ADC_12_Control .....	238
6)	R_ADC_12_Read .....	239
4.2.22.	10-bit Analog to Digital Converter .....	241
1)	R_ADC_10_Create .....	241
2)	R_ADC_10_Destroy .....	246
3)	R_ADC_10_Control .....	247
4)	R_ADC_10_Read .....	248
5.	Usage Examples .....	249
5.1.	Interrupt control .....	249
5.2.	I/O Port .....	251
5.3.	Voltage Detection Circuit .....	253
5.4.	Bus Controller .....	254
5.5.	Data Transfer Controller .....	255
5.5.1.	Block transfer mode .....	255
5.5.2.	Chain transfer operation .....	258
5.6.	General PWM Timer Driver .....	261
5.7.	Compare Match Timer .....	262
5.8.	Independent Watchdog Timer .....	263
5.9.	Serial Communication Interface .....	264
5.9.1.	SCI Reception .....	264
5.9.2.	SCI Transmission .....	266
5.9.3.	Synchronous Transmission and Reception .....	268
5.9.4.	SCI Reception in Asynchronous Multi-Processor mode .....	270
5.9.5.	SCI Transmission in Asynchronous Multi-Processor mode .....	272
5.10.	CRC calculator .....	274
5.11.	I <sup>2</sup> C Bus Interface .....	275
5.11.1.	Master mode .....	275

1)	Configuration and transmission .....	276
2)	Reception .....	278
3)	Repeated Start .....	279
5.11.2.	Master mode with DTC .....	280
5.11.3.	Slave mode .....	284
5.12.	Serial Peripheral Interface .....	286
5.12.1.	Master operation with multiple slaves .....	286
5.13.	LIN module .....	289
5.13.1.	Transmit LIN data .....	290
5.13.2.	Receive data from a LIN slave .....	291
5.13.3.	Self-test mode .....	293
5.14.	12-bit Analog to Digital Converter .....	296
5.15.	10-bit Analog to Digital Converter .....	298
6.	RX-specific notes .....	299
6.1.	Interrupts and processor mode .....	299
6.2.	Interrupts and DSP instructions .....	299
<b>Revision History .....</b>		<b>1</b>

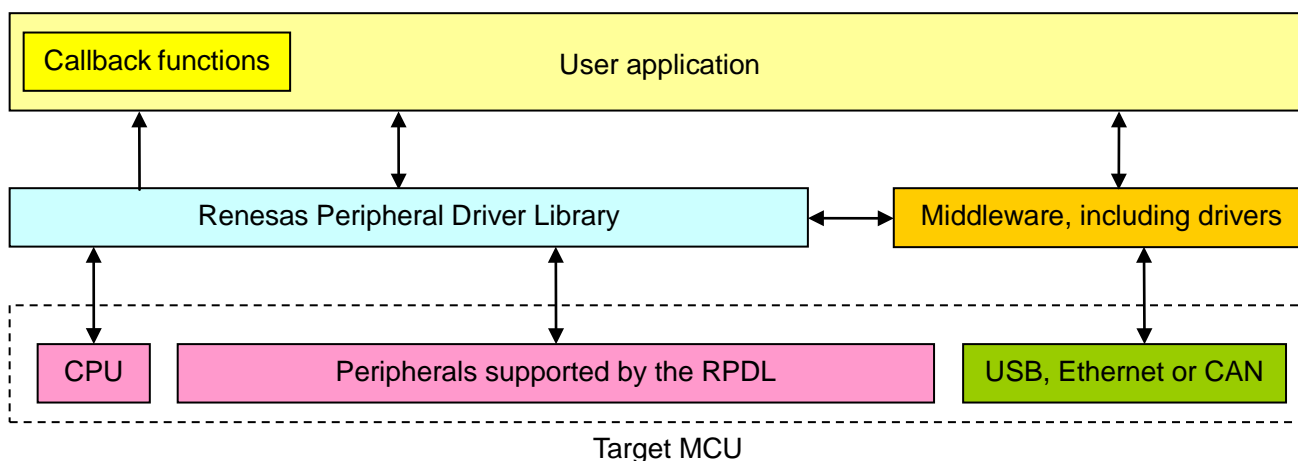


## 1. Introduction

The Renesas Peripheral Driver Library (RPDL) is a unified API for controlling the peripheral modules on the microcontrollers made by Renesas Electronics.



**Figure 1-1: System configuration, with all peripherals supported by RPDL**



**Figure 1-2: System configuration, with middleware taking direct control of some peripherals**

The library is packaged as:

- a) A binary file containing all of the peripheral driver functions,
- b) Header files containing the information that the user needs to call any of the functions from their own application code and
- c) Interrupt handlers supplied as source code.

For best use of this library, it is required that the user will have the following documents as a minimum:

- i. The schematic
- ii. The MCU hardware manual
- iii. This RPDL API User's manual

The binary file is produced using the Renesas RX C compiler. It should be usable by another compiler that conforms to the Renesas Application Binary Interface.

The coding standards and naming conventions are specified by Renesas.

### 1.1. Tool chain requirements

This RPDG library has been built and tested using the C/C++ Compiler Package for RX Family V.1.02 Release 01. It cannot be used with older versions of the tool chain.

The latest version of the tool chain can be downloaded from the Renesas Web site ([Home / Products / Software and Tools / Coding Tools / C/C++ Compilers and Assemblers / C/C++ Compiler Package for RX Family /](#)).

### 1.2. Using the library within your project

The driver library can be used in two ways.

#### 1.2.1. Via the PDG graphical utility

PDG can be downloaded from [www.renesas.com/pdg](http://www.renesas.com/pdg).

The directions for use of the PDG utility are given in the PDG manual.

#### 1.2.2. Added to a project by the user and used stand-alone

To add the driver library to your project's build environment, you need to

- a) Unzip the RPDG distribution.
- b) Copy the required source, header and library files into your project folder.
- c) Include the required source files.
- d) Add the driver library file to the linked files list.

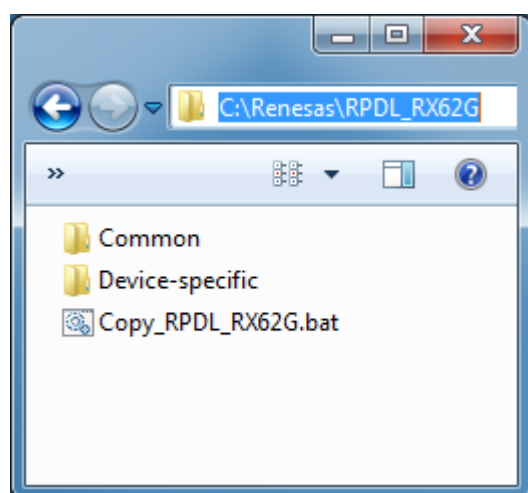
The instructions to follow for stand-alone use start are given below.

##### 1) Unzip the RPDG files

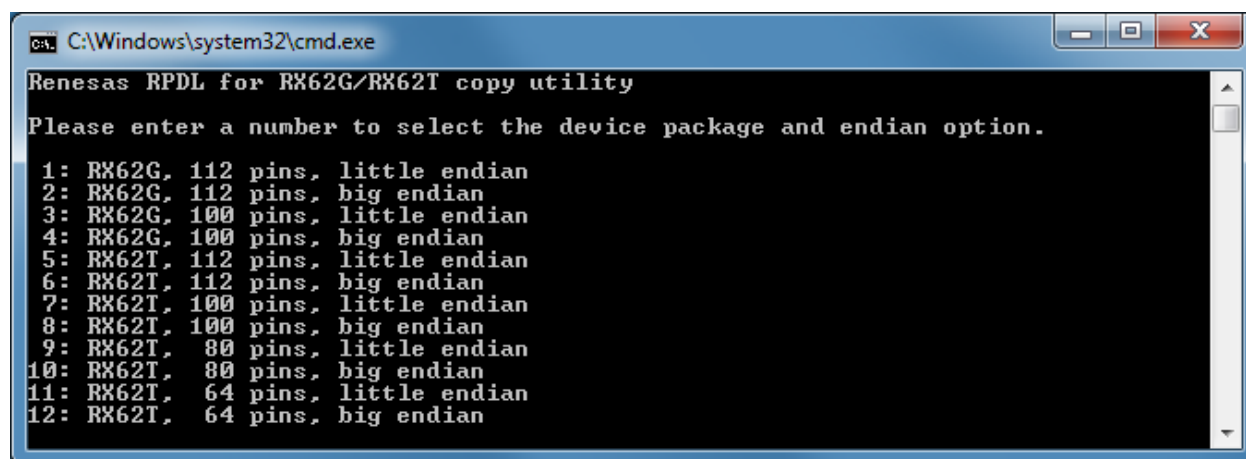
Double-click on the file RPDG\_RX62G.exe to unpack the files.  
The default location is C:\Renesas\RPDG\_RX62G.

##### 2) Copy the files into your project area

Navigate to where the RPDG files were unpacked.

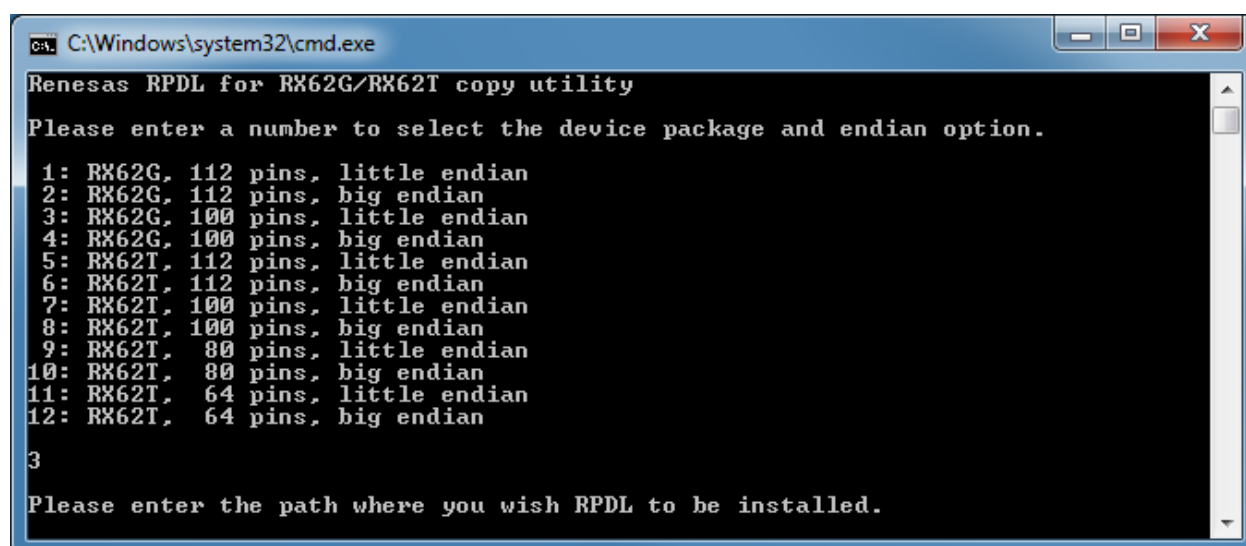


Double-click on "Copy\_RPDG\_RX62G.bat" to start the copy process.



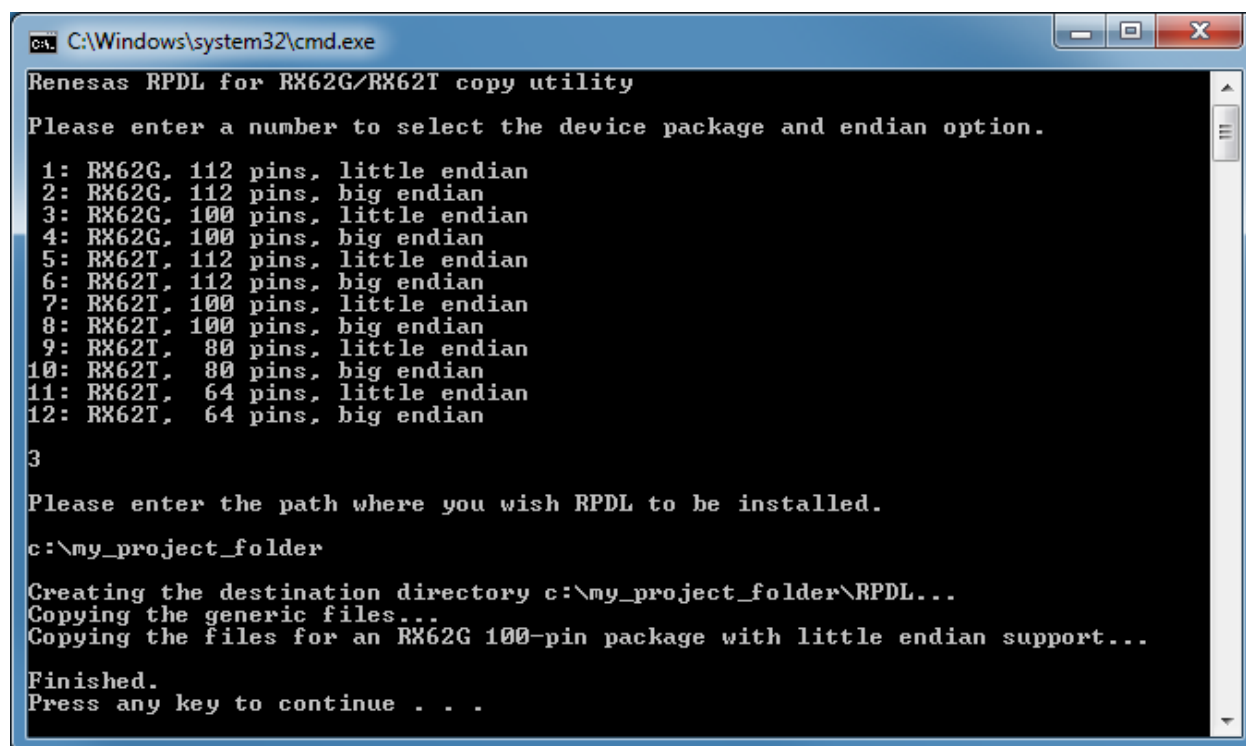
```
C:\Windows\system32\cmd.exe
Renesas RPD L for RX62G/RX62T copy utility
Please enter a number to select the device package and endian option.
1: RX62G, 112 pins, little endian
2: RX62G, 112 pins, big endian
3: RX62G, 100 pins, little endian
4: RX62G, 100 pins, big endian
5: RX62T, 112 pins, little endian
6: RX62T, 112 pins, big endian
7: RX62T, 100 pins, little endian
8: RX62T, 100 pins, big endian
9: RX62T, 80 pins, little endian
10: RX62T, 80 pins, big endian
11: RX62T, 64 pins, little endian
12: RX62T, 64 pins, big endian
```

Select the device package option by pressing a number, and then press Enter.



```
C:\Windows\system32\cmd.exe
Renesas RPD L for RX62G/RX62T copy utility
Please enter a number to select the device package and endian option.
1: RX62G, 112 pins, little endian
2: RX62G, 112 pins, big endian
3: RX62G, 100 pins, little endian
4: RX62G, 100 pins, big endian
5: RX62T, 112 pins, little endian
6: RX62T, 112 pins, big endian
7: RX62T, 100 pins, little endian
8: RX62T, 100 pins, big endian
9: RX62T, 80 pins, little endian
10: RX62T, 80 pins, big endian
11: RX62T, 64 pins, little endian
12: RX62T, 64 pins, big endian
3
Please enter the path where you wish RPD L to be installed.
```

Type the full path to the folder where you wish RPD L to be copied to, and then press Enter.  
The utility will create a folder in the location that you specified and copy the files into the new folder.



```
C:\Windows\system32\cmd.exe
Renesas RPD L for RX62G/RX62T copy utility
Please enter a number to select the device package and endian option.
1: RX62G, 112 pins, little endian
2: RX62G, 112 pins, big endian
3: RX62G, 100 pins, little endian
4: RX62G, 100 pins, big endian
5: RX62T, 112 pins, little endian
6: RX62T, 112 pins, big endian
7: RX62T, 100 pins, little endian
8: RX62T, 100 pins, big endian
9: RX62T, 80 pins, little endian
10: RX62T, 80 pins, big endian
11: RX62T, 64 pins, little endian
12: RX62T, 64 pins, big endian
3
Please enter the path where you wish RPD L to be installed.
c:\my_project_folder
Creating the destination directory c:\my_project_folder\RPD L...
Copying the generic files...
Copying the files for an RX62G 100-pin package with little endian support...
Finished.
Press any key to continue . . .
```

Press any key to close the window.

## 3) Include the new directory

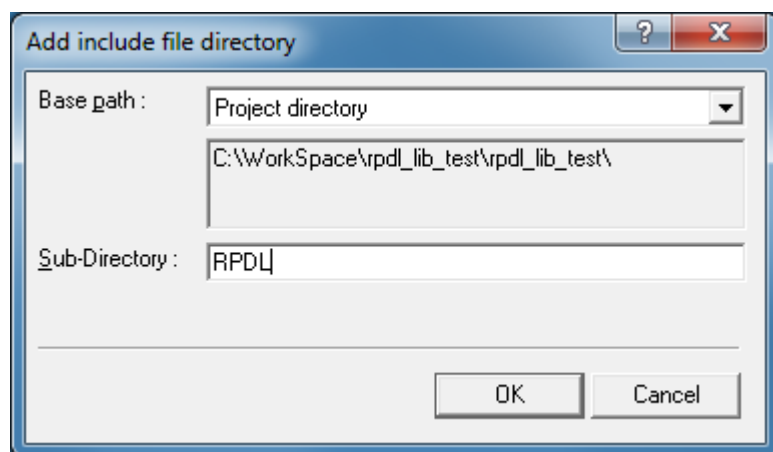
Use the key sequence Alt, B, R to open the “RX Standard Toolchain” window.

Select the C/C++ tab.

Use the key sequence S, I to show the included file directories.

Click on the “Add...” button.

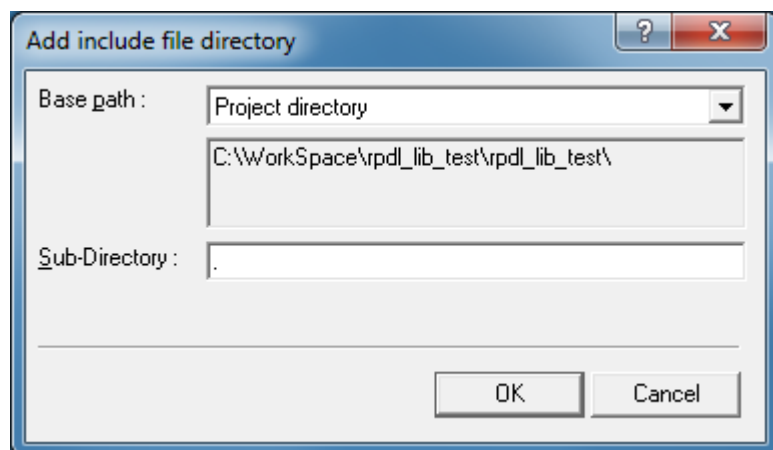
In the “Add include file directory” window, enter the details as shown:



Click on “OK” to close the window.

Click on the “Add...” button.

In the “Add include file directory” window, enter the details as shown:



Click on “OK” to close the window.

## 4) Add the RPD library file

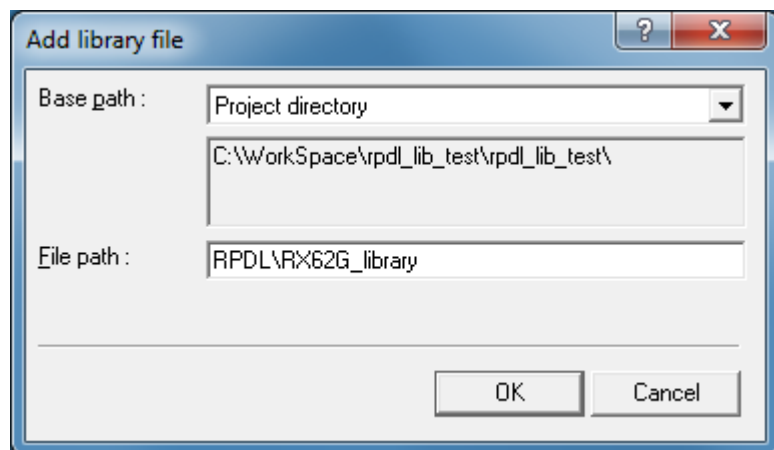
The library file is added to the list used by the linker application.

Select the Link/Library tab.

From the “Show entries for :” drop-down menu, select “Library files”.

Click on the “Add...” button.

In the “Add library file” window, select “Project directory” and enter “RPDL\RX62G\_library”.



Click on “OK” to close the window.

Click on “OK” to return to the main HEW window.

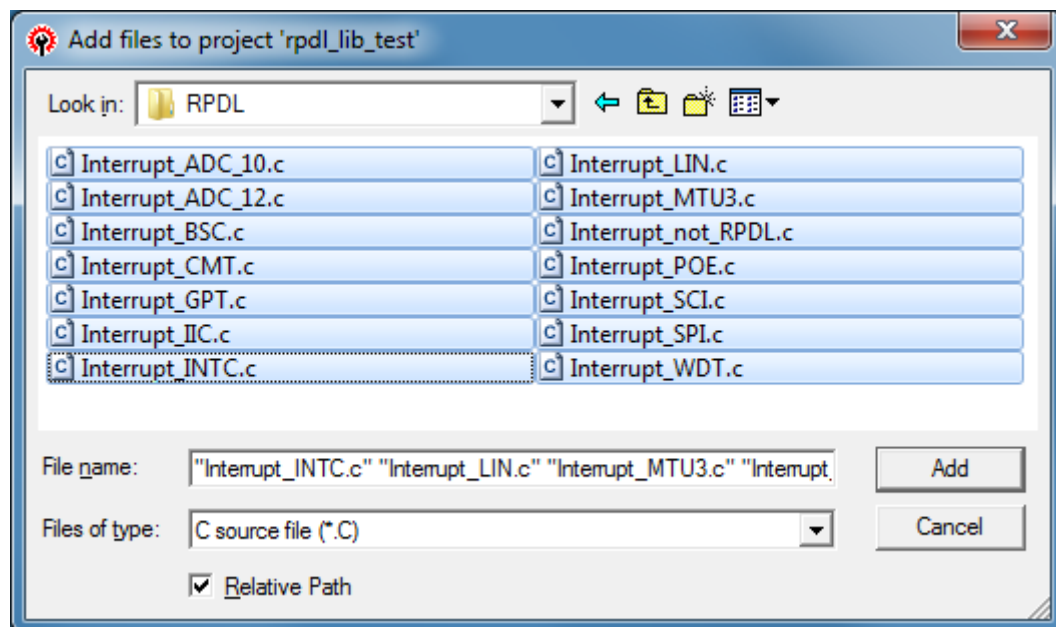
## 5) Include the new source files

Use the key sequence Alt, P, A to open the “Add files to project ‘<your project>’” window.

Double click on the RPD\_L folder.

From the “Files of type” drop-down list, select “C source file (\*.C)”.

Use the key sequence Ctrl-A to select all of the files, as shown below.



Click on “Add”.

Click on “OK” to return to the main HEW window.

## 6) Peripherals that are not required

If a peripheral module is not required, the interrupt handler file does not need to be included.

If the unused interrupts still require entries in the interrupt vector table, edit the file `Interrupt_not_RPDL.c` to uncomment the `#define` for the unused peripherals.

For example,

```
//#define RPDL_ADC_12_not_used
```

Becomes

```
#define RPDL_ADC_12_not_used
```

The file `Interrupt_INTC.c` must be included.

## 7) Peripherals that are not supported by RPDL

The file `Interrupt_not_RPDL.c` also contains handlers for the peripherals that are not supported by RPDL. This allows the user to add handler code for these peripherals while supporting the Fast Interrupt feature (see `R_INTC_CreateFastInterrupt`).



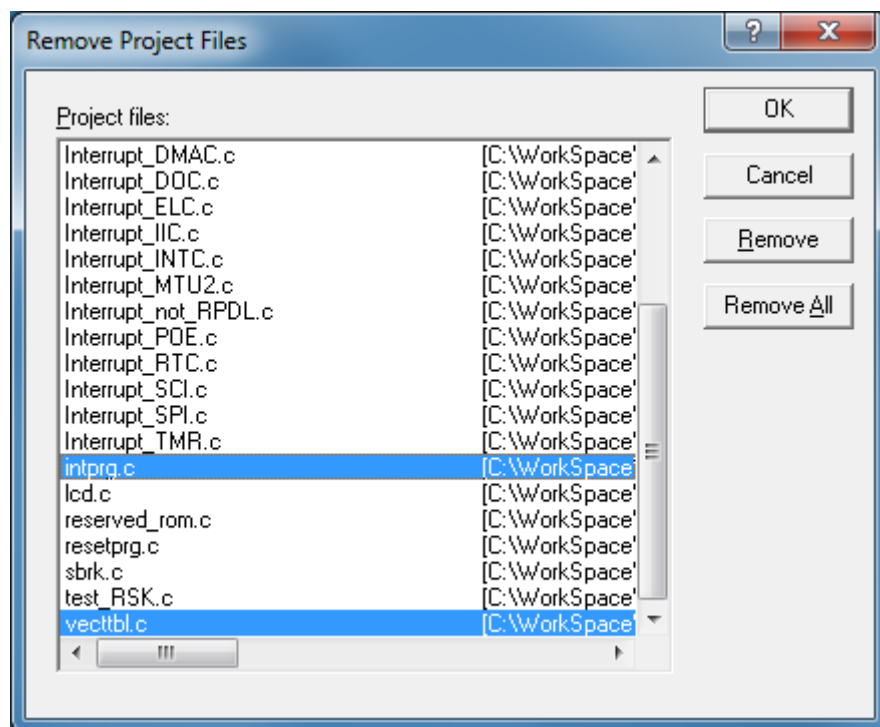
## 8) Avoid conflicts with standard project files

If the files 'intprg.c' or 'vecttbl.c' are included in the project, remove or exclude them.

## (a) Removal

Use the key sequence Alt, P, R to open the "Remove Project Files" window.

Select the files and click on Remove.



## (b) Exclusion

Select the two files and use the key sequence Alt, B, I to exclude them.

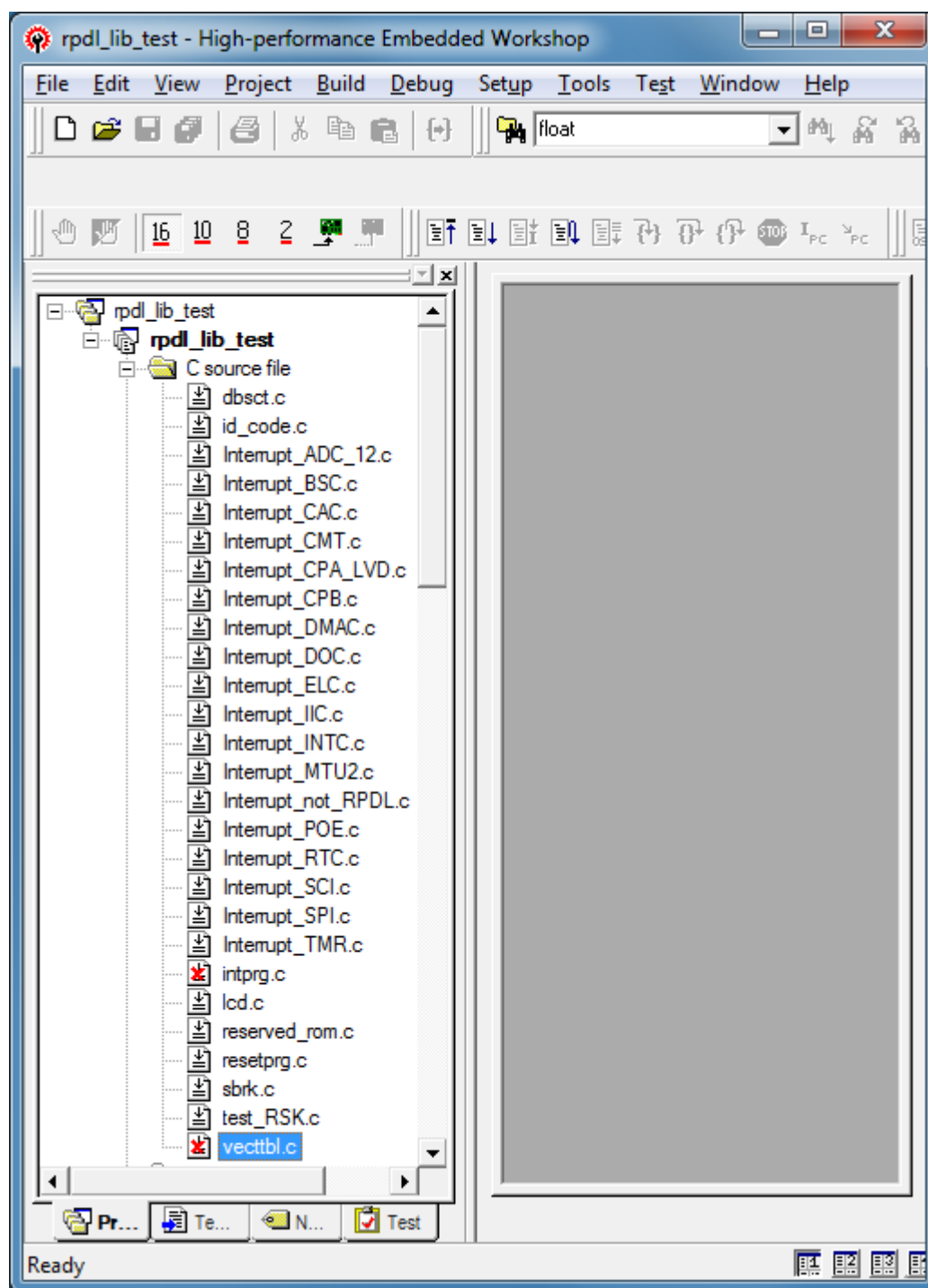


Figure 1-3: intprg.c and vecttbl.c have been excluded

## 9) Set the build options.

Use the key sequence Alt, B, R to open the "RX Standard Toolchain" window.

### (a) Set the optimisation

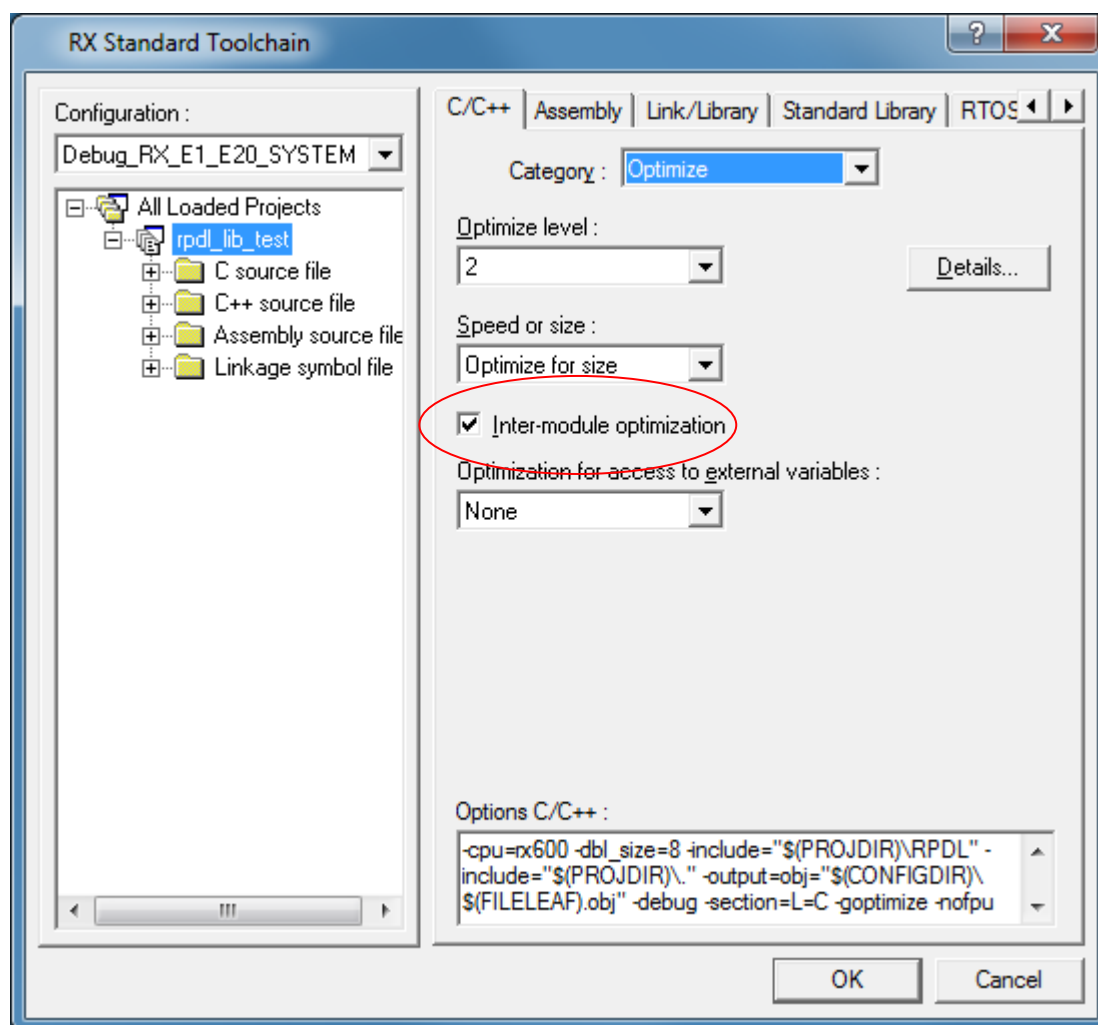
To avoid linking unused RPD functions, adjust the Compiler and Linker settings.

#### (i) Compiler

Select the C/C++ tab.

Use the key sequence Y, O, O to show the optimisation options.

Ensure that the "Inter-module optimization" option is enabled.

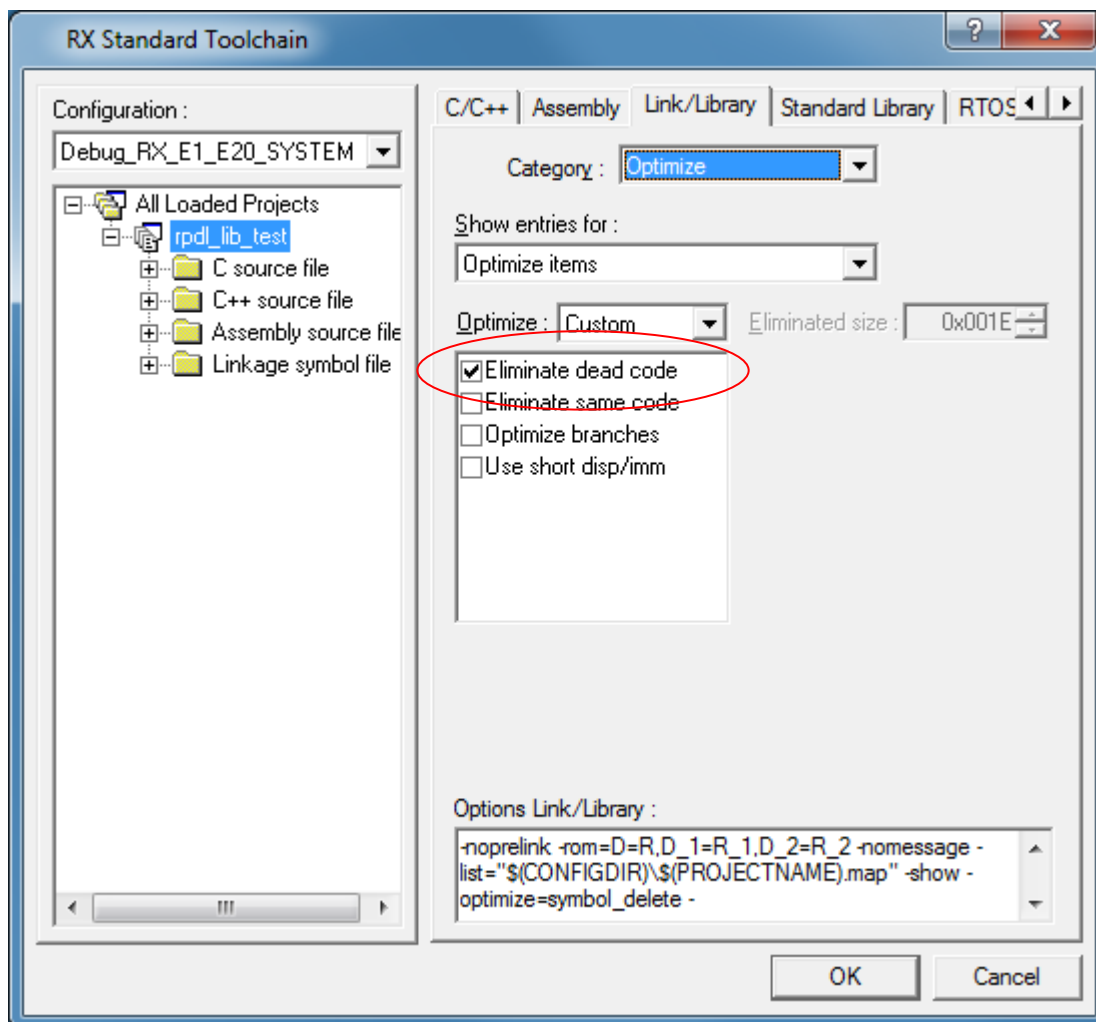


**(ii) Linker**

Select the Link/Library tab.

Use the key sequence Y, O, O to show the optimisation options.

If the "Eliminate dead code" option is not enabled, from the Optimize drop-down list select Custom and enable the option.



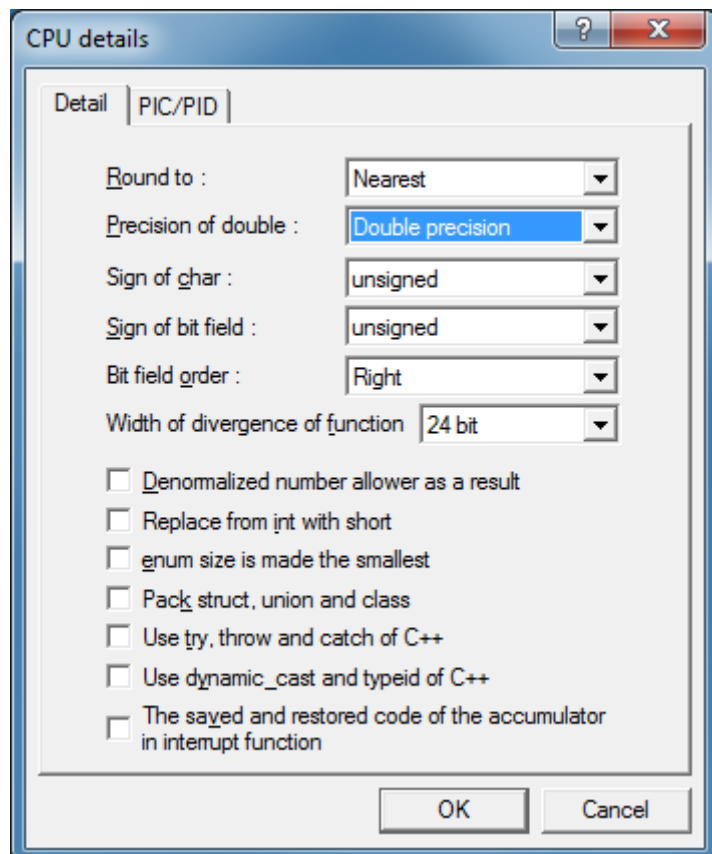
### (b) Set the floating point precision

The wide range of possible internal clock frequencies requires double-precision floating point number storage.

Select the CPU tab.

Click on the Details... button to open the “CPU details” window.

Use the drop-down menu to select Double precision.



Click on “OK” to close the window.

Click on “OK” to return to the main HEW window.

## 10) Build the project

No further configuration should be required.  
Simply build the project.

### 1.2.3. Header file inclusion

The RPD\_L folder contains a header file, `iodefne_RPD_L.h`.

This file is included by the RPD\_L source files and will also be included by any user-generated files that call RPD\_L functions.

The main HEW project folder may contain the header file `iodefne.h`.

This file is normally used if access to the I/O registers in the MCU is required.

For any user-generated files that call RPD\_L functions, there is no need to include this file `iodefne.h`.

### 1.2.4. Header file order

The file `r_pdl_definitions.h` must be included after any peripheral-specific header file.

For example:

```
/* Peripheral driver function prototypes and definitions */
#include "r_pdl_cgc.h"
#include "r_pdl_cmt.h"

/* PDL device-specific definitions */
#include "r_pdl_definitions.h"
```

### 1.3. Document structure

The drivers are summarised in section 2 and explained in detail in section 4.

Section 5 provides usage examples.

Section 6 provides details which are specific to the RX CPU.

## 1.4. List of Abbreviations and Acronyms

ADC	Analog to Digital Converter
API	Application Programming Interface
BCD	Binary-Coded Decimal
Bit	Binary digit
BSC	Bus State Controller
CAN	Controller Area Network
CMT	Compare Match Timer
CGC	Clock Generation Circuit
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DAC	Digital to Analog Converter
DC	Direct Current
DSP	Digital Signal Processing
DTC	Data Transfer Controller
EEPROM	Electrically Erasable and Programmable ROM
FIFO	First-In, First-Out
GSM	Global System for Mobile communications
HEW	High-performance Embedded Workbench
I <sup>2</sup> C	Inter-Integrated Circuit
INTC	Interrupt Controller
I/O	Input / Output
kB	Kilo Byte (1024 bytes)
LIN	Local Interconnect Network
LOCO	Low-speed On-Chip Oscillator
LPC	Low Power Consumption
LSB	Least-Significant Bit
MCU	Microcontroller Unit
MTU	Multi-function Timer pulse Unit
NMI	Non-Maskable Interrupt
MSB	Most-Significant Bit
PDG	Peripheral Driver Generator
PFC	Port Function Control
POE	Port Output Enable
PWM	Pulse-Width Modulation
RAM	Random-Access Memory
ROM	Read-Only Memory
RPDL	Renesas Peripheral Driver Library
RSPI	Renesas SPI
SCI	Serial Communications Interface
SMBus	System Management Bus
SPI	Serial Peripheral Interface
WDT	Watchdog Timer

All trademarks and registered trademarks are the property of their respective owners.



## 2. Driver

### 2.1. Overview

This library provides a set of peripheral function control programs (peripheral drivers) for Renesas microcontrollers and allows the peripheral driver to be built into a user program.

### 2.2. Control Functions summary

This library has the following control functions available as peripheral drivers.

- (1) Clock Generation Circuit  
These driver functions are used to configure the multiple internal clock signals.
- (2) Interrupt  
These driver functions are used for configuring the external interrupt pins, handling fixed interrupts and controlling the interrupt priority.
- (3) I/O Port  
These driver functions are used to configure the I/O pins and provide data read, write, compare and modify operations.
- (4) Port Function  
These driver functions are used for configuring the I/O pin optional functions.
- (5) MCU Operation  
These driver functions are used for configuring the MCU operation.
- (6) Low Power Consumption  
These driver functions are used for selecting lower power consumption.
- (7) Voltage Detection  
The driver function supports access to the registers, which control the voltage detection circuit.
- (8) Bus Controller  
These driver functions are used for configuring the external address bus, data bus and chip select pins and handling any bus errors.
- (9) Data Transfer Controller  
These driver functions are used for configuring and controlling the transfer of data triggered by peripheral interrupts.
- (10) Multi-Function Timer Pulse Unit  
These driver functions are used for configuring and controlling the timers.
- (11) Port Output Enable  
The driver functions support the use of the Port Output module.
- (12) General PWM Timer  
The driver functions support the use of the four 16-bit PWM timers.
- (13) Compare Match Timer  
These driver functions are used for configuring and controlling the timers.
- (14) Watchdog Timer  
These driver functions are used for configuring and controlling the timer.
- (15) Independent Watchdog Timer  
The driver functions support the use of the independent watchdog timer.

(16) Serial Communication Interface

These driver functions are used to configure the serial channels and manage the transmission and / or reception of data across them.

(17) CRC calculator

These driver functions are used for controlling the calculator.

(18) I<sup>2</sup>C Bus Interface

These driver functions are used for controlling the I<sup>2</sup>C bus channels.

(19) Serial Peripheral Interface

These driver functions are used to configure the SPI channel and manage the transmission and / or reception of data.

(20) LIN module

The driver functions support the use of the LIN serial channel.

(21) Analog to Digital Converter

These driver functions are used for configuring the ADC units, controlling the units and reading the conversion results.

(22) Digital to Analog converter

These driver functions are used for configuring the DAC module and setting the output voltages.

### 2.3. Clock Generation Circuit Driver

The driver functions support the control of the internal clock generator, providing the following operations.

1. Configuration of the multiple clock outputs for system, peripheral and external bus operation.
2. Reading the Clock generator status flags.

Note: Configuring the Clock Generation Circuit also provides information on clock frequencies that will be used by the integrated drivers for other peripherals.

## 2.4. Interrupt Control Driver

The driver functions support the use of the interrupt controller, providing the following operations.

1. Configuration an external interrupt pin for use.
2. Assigning an interrupt to be processed using the Fast Interrupt route.
3. Assigning handlers for the fixed exception interrupts.
4. Controlling an external interrupt input.
5. Reading the status of an external interrupt.
6. Reading an interrupt register.
7. Writing to an interrupt register.
8. Modifying an interrupt register.

## 2.5. I/O Port Driver

The driver functions support the use of the I/O port pins, providing the following operations.

1. Configuration for use.
2. Reading the pin or port configuration.
3. Modifying the pin or port configuration.
4. Reading a pin or 8-bit port value.
5. Writing to a pin or 8-bit port.
6. Comparing a pin or 8-bit port with a supplied value.
7. Modifying a pin or 8-bit port using a logical operation.
8. Waiting until a pin or 8-bit port matches a supplied value.

## 2.6. Port Function Control Driver

The driver functions support access to the Port Function Control (PFC) registers which select the mode of operation for some I/O pins.

The other driver functions modify the PFC registers automatically. For peripherals that are not supported by the driver library, these functions support:

1. Reading from a PFC register.
2. Writing to a PFC register.
3. Modifying a PFC register

## 2.7. MCU Operation Driver

The driver functions support access to the registers which select the mode of operation for the microcontroller. These functions support:

1. Controlling the on-chip ROM and RAM.
2. Reading the MCU status flags.

## 2.8. Low Power Consumption Driver

The driver functions support access to the registers which select the lower power modes of operation for the microcontroller. These functions support:

1. Configuring the state while in standby mode, and the activity that can be used to resume operation.
2. Selecting one of the low-power modes.
3. Writing data to the backup memory area.
4. Reading data from the backup memory area.
5. Determining the cause of the exit from the lowest power mode.



## 2.9. Voltage Detection Circuit Driver

The driver function supports access to the registers, which control the voltage detection circuit. This function supports:

1. Configuring the response to the supply voltage dropping below either voltage threshold.

### 2.10. Bus Controller Driver

The driver functions support the control of the internal bus, providing the following operations.

1. Configuration of the controller.
2. Controlling the bus controller.
3. Reading the status of the controller.

### 2.11. Data Transfer Controller Driver

The driver functions support the control of the Data Transfer Controller, providing the following operations.

1. Setting the central options.
2. Configuration for use, including support for chain transfers.
3. Disabling the controller.
4. Starting or stopping the controller.
5. Reading the status flags and data transfer registers.

## 2.12. Multi-Function Timer Pulse Unit Driver

The driver functions support the use of the seven 16-bit timers, providing the following operations.

1. Selection of the MTU pins for use.
2. Configuration for use, including
  - Access to all control bits.
  - Automatic interrupt control
  - Automatic I/O pin configuration
3. Disabling channels that are no longer required and enabling low-power mode.
4. Control of a timer channel.
5. Control of a timer unit.
6. Reading the status flags and registers of a timer channel.
7. Reading the status flags and registers of a timer unit.

Note: The Clock Generation Circuit must be configured before configuring any timer channel.

### 2.13. Port Output Enable Driver

The driver functions support the use of the Port Output module, providing the following operations.

1. Configuring the pins for use.
2. Configuring the interrupts and callback functions.
3. Run-time control of outputs, interrupts and flags.
4. Checking the module status.

## 2.14. General PWM Timer Driver

The driver functions support the use of the four 16-bit PWM timers, providing the following operations.

1. Selection of the GPT pins for use, and configuring the LOCO count module.
2. Configuration for use, including
  - Access to all control bits.
  - Automatic interrupt control
  - Automatic I/O pin configuration
3. Disabling channels that are no longer required and enabling low-power mode.
4. Control of a timer channel.
5. Control of a timer unit.
6. Reading the status flags and registers of a timer channel.
7. Reading the status flags and registers of a timer unit.

### 2.15. Compare Match Timer Driver

The driver functions support the use of the two 16-bit timers, providing the following operations.

1. Configuration for use, including
  - Automatic clock setting using frequency or period as an input.
  - Manual clock setting using register values as inputs.
  - Automatic interrupt control
2. Configuration for use as a one-shot timer.
3. Disabling channels that are no longer required and enabling low-power mode.
4. Control of a timer, including constant register updates, change of frequency.
5. Reading the counter value and status flag.

Note: The Clock Generation Circuit must be configured before configuring any timer channel.

## 2.16. Watchdog Timer Driver

The driver functions support the use of the watchdog timer, providing the following operations.

1. Configuring the timer for use, including
  - Automatic clock setting using frequency or period as an input.
  - Internal timer mode
  - Watchdog timer mode
  - MCU reset generation while in watchdog timer mode
  - Automatic interrupt control
2. Control of the timer, including
  - Stopping the timer
  - Counter refresh to prevent overflow
3. Reading the timer status and counter register.

Note: The Clock Generation Circuit should be configured before configuring this timer.



### 2.17. Independent Watchdog Timer Driver

The driver functions support the use of the independent watchdog timer, providing the following operations.

1. Configuring the timer for use.
2. Refreshing the timer to prevent the reset operation.
3. Reading the timer status and counter register.

### 2.18. Serial Communication Interface Driver

The driver functions support the use of the seven serial communication channels, providing the following operations.

1. Selecting the pins to be used.
2. Configuration for use, including
  - Automatic baud rate clock calculations
  - Automatic interrupt control
  - Automatic I/O pin configuration
3. Disabling channels that are no longer required and enabling low-power mode.
4. Transmitting data, with polling or interrupt mode automatically selected.
5. Receiving data, with polling or interrupt mode automatically selected.
6. Control the channel operation.
7. Reading the status flags.

Note: The Clock Generation Circuit must be configured before configuring any serial channel.

### 2.19. CRC Calculator Driver

The driver functions support the CRC calculator, providing the following operations.

1. Configuration for use, including
  - Polynomial selection.
  - Bit order selection.
  - Preparation for a new calculation.
2. Disabling the calculator and enabling low-power mode.
3. Writing data to be used for the calculation.
4. Reading the calculation result.

## 2.20. I<sup>2</sup>C Bus Interface Driver

The driver functions support the use of the I<sup>2</sup>C modules, providing the following operations.

1. Configuration for use, including
  - Automatic clock setting using transfer rate as an input.
  - Automatic interrupt control
  - Automatic I/O pin configuration
2. Disabling modules that are no longer required and enabling low-power mode.
3. Transmitting data in Master mode.
4. Receiving data in Master mode.
5. Completing the reception of data in Master mode.
6. Monitoring the bus and handling the reception of data in Slave mode.
7. Transmitting data in Slave mode.
8. Control of one or more units, including bus lock-up recovery support.
9. Reading the status of a module.

Note: The Clock Generation Circuit must be configured before configuring any I<sup>2</sup>C module.

## 2.21. Serial Peripheral Interface Driver

The driver functions support the use of one SPI channel, providing the following operations.

1. Configuration for use, including
  - Automatic clock setting using transfer rate as an input.
  - Automatic I/O pin configuration
2. Disabling channels that are no longer required and enabling low-power mode.
3. Configuration of command sequence settings.
4. Managing the transfer of data on the interface, including
  - Automatic interrupt control
  - Automatic DTC control.
5. Control of special modes such as loopback.
6. Reading the status of a module.

Note: The Clock Generation Circuit must be configured before configuring any SPI channel.

## 2.22. LIN module

The driver functions support the use of the LIN channel, providing the following operations.

1. Configuration for use, including
  - Automatic clock setting using the transfer rate as an input.
  - Automatic I/O pin configuration
2. Disabling the channel when no longer required and enabling low-power mode.
3. Managing the transfer of data on the LIN bus, including
  - Automatic interrupt control
4. Reading data received from a LIN slave device.
5. Control of special modes such as self-test.
6. Reading the status of the LIN module.

Note: The Clock Generation Circuit must be configured before configuring the LIN channel.

---

### 2.23. 12-bit Analog to Digital Converter Driver

The driver functions support the use of the 12-bit ADC unit, providing the following operations.

1. I/O pin configuration
2. Unit specific configuration for use, including
  - Automatic clock setting using sampling time as an input
  - Automatic interrupt control
3. Channel specific configuration for use, including
  - Comparator control
  - Gain amplifier control
4. Disabling the unit when no longer required and enabling low-power mode.
5. Control of one or more units, including
  - CPU sleep option
6. Reading the conversion results, with support for polling or interrupts.

Note: The Clock Generation Circuit must be configured before configuring the ADC unit.

## 2.24. 10-bit Analog to Digital Converter Driver

The driver functions support the use of the ADC unit, providing the following operations.

1. Configuration for use, including
  - Automatic clock setting using sampling time as an input.
  - Automatic interrupt control
  - Automatic I/O pin configuration
2. Disabling ADC unit when it is no longer required, and enabling low-power mode.
3. Control ADC unit, including
  - CPU sleep option
4. Reading the conversion results of ADC unit, with support for polling or interrupts.

Note: The Clock Generation Circuit must be configured before configuring any ADC unit.



### 3. Types and definitions

#### 3.1. Data types

This section describes the data types used in this library. For details about the setting values, refer to the section "4.2 Description of Each API".

The header files `stdint.h` and `stdbool.h` are included with the Renesas RX compiler.

**Table 1: Data types**

Type	Defined in	Description	Range
<code>bool</code>	<code>stdbool.h</code>	Boolean	0 (false) to 1 (true)
<code>float</code>	C	Floating point, 32 bits	$\pm\infty$
<code>uint8_t</code>	<code>stdint.h</code>	Unsigned, 8 bits	0 to 255
<code>uint16_t</code>		Unsigned, 16 bits	0 to $2^{16} - 1$
<code>uint32_t</code>		Unsigned, 32 bits	0 to $2^{32} - 1$

#### 3.2. General definitions

##### 3.2.1. PDL\_NO\_FUNC

Used as a parameter when there is no applicable function.

##### 3.2.2. PDL\_NO\_PTR

Used as a parameter when there is no applicable data location.

##### 3.2.3. PDL\_NO\_DATA

Used as a parameter when there is no applicable data value.

##### 3.2.4. PDL\_MCU\_GROUP

The family supported by this build of the driver library. It is defined as RX62G or RX62T, depending on the group selected when RPD\_L was installed.

A usage example is:

```
#if PDL_MCU_GROUP != RX62T
#error "Wrong RPD_L ! "
#endif
```

##### 3.2.5. PDL\_VERSION

The version number of the RPD\_L library. The number is stored in BCD format (xx.xx). For example, 0100h is v1.00.

A usage example is:

```
const uint16_t rpd_l_version_number = PDL_VERSION;
```

##### 3.2.6. Bit definitions

The definitions `BIT_n` and `INV_BIT_n`, where `n` = 0 to 31, are available to the user.

## 4. Library Reference

### 4.1. API List by Peripheral Function

Table 4.1 lists the Renesas Embedded APIs by peripheral function.

**Table 4.1 Renesas Embedded API List**

Category	Num.	Name	Description
Clock Generation Circuit	1	R_CGC_Set	Configure the clock generation circuit.
	2	R_CGC_GetStatus	Read the clock status register.
Interrupt control unit	1	R_INTC_CreateExtInterrupt	Configure an external interrupt pin.
	2	R_INTC_CreateSoftwareInterrupt	Enable use of the software interrupt.
	3	R_INTC_CreateFastInterrupt	Assign handlers for the fixed-vector interrupts.
	4	R_INTC_CreateExceptionHandler	Enable faster interrupt processing for one interrupt.
	5	R_INTC_ControlExtInterrupt	External interrupt control.
	6	R_INTC_GetExtInterruptStatus	Read the external interrupt status.
	7	R_INTC_Read	Read an interrupt register.
	8	R_INTC_Write	Update an interrupt register.
	9	R_INTC_Modify	Modify an interrupt register.
I/O port	1	R_IO_PORT_Set	Configure an I/O port.
	2	R_IO_PORT_ReadControl	Read an I/O port's control registers.
	3	R_IO_PORT_ModifyControl	Modify an I/O port's control registers.
	4	R_IO_PORT_Read	Read data from an I/O port.
	5	R_IO_PORT_Write	Write data to an I/O port.
	6	R_IO_PORT_Compare	Check the pin states on an I/O port.
	7	R_IO_PORT_Modify	Modify the pin states on an I/O port.
	8	R_IO_PORT_Wait	Wait for a match on an I/O port.
Port Function Control	1	R_PFC_Read	Read a PFC register.
	2	R_PFC_Write	Write to a PFC register.
	3	R_PFC_Modify	Modify a PFC register.
MCU operation	1	R_MCU_Control	Control the operation of the MCU.
	2	R_MCU_GetStatus	Read the MCU status.
Low Power Consumption	1	R_LPC_Create	Configure the MCU low power conditions.
	2	R_LPC_Control	Select a low power consumption mode.
	3	R_LPC_WriteBackup	Write to the Backup registers.
	4	R_LPC_ReadBackup	Read from the Backup registers.
	5	R_LPC_GetStatus	Read the status flags.
Voltage Detection Circuit	1	R_LVD_Control	Configure the voltage detection circuit.
Bus Controller	1	R_BSC_Create	Configure the bus controller.
	2	R_BSC_Control	Modify the Bus Controller operation.
	3	R_BSC_GetStatus	Read the Bus Controller status flags.
Data Transfer Controller	1	R_DTC_Set	Set the Data Transfer Controller options.
	2	R_DTC_Create	Configure the DTC for a transfer.
	3	R_DTC_Destroy	Shutdown the Data Transfer Controller.
	4	R_DTC_Control	Control the Data Transfer Controller.
	5	R_DTC_GetStatus	Check the status of the Data Transfer Controller.
Multi-Function Timer Pulse Unit	1	R_MTU3_Set	Configure the Multi-function Timer Pulse Unit.
	2	R_MTU3_Create	Configure an MTU channel.
	3	R_MTU3_Destroy	Disable the Multi-function Timer Pulse Unit.
	4	R_MTU3_ControlChannel	Control an MTU channel.
	5	R_MTU3_ControlUnit	Control the Multi-function Timer Pulse Unit.
	6	R_MTU3_ReadChannel	Read from MTU channel registers.
	7	R_MTU3_ReadUnit	Read from MTU unit registers.
Port Output Enable	1	R_POE_Set	Configure the Port Output Enable module.
	2	R_POE_Create	Configure the Port Output Enable event handling.
	3	R_POE_Control	Control the Port Output Enable module.
	4	R_POE_GetStatus	Check the status of the Port Output Enable module.

General PWM Timer	1	R_GPT_Set	Configure the GPT unit.
	2	R_GPT_Create	Configure a GPT channel.
	3	R_GPT_Destroy	Disable the GPT unit.
	4	R_GPT_ControlChannel	Control a GPT channel.
	5	R_GPT_ControlUnit	Control the GPT unit.
	6	R_GPT_ReadChannel	Read from GPT channel registers.
	7	R_GPT_ReadUnit	Read the GPT unit flags.
Compare Match Timer	1	R_CMT_Create	Configure a CMT channel.
	2	R_CMT_CreateOneShot	Configure a CMT channel as a one-shot event.
	3	R_CMT_Destroy	Disable a CMT unit.
	4	R_CMT_Control	Control CMT operation.
	5	R_CMT_Read	Read CMT channel status and registers.
Watchdog Timer	1	R_WDT_Create	Configure the Watchdog timer.
	2	R_WDT_Control	Control the Watchdog operation.
	3	R_WDT_Read	Read the Watchdog timer status and registers.
Independent Watchdog Timer	1	R_IWDT_Set	Configure the Independent Watchdog operation.
	2	R_IWDT_Control	Control the Independent Watchdog operation.
	3	R_IWDT_Read	Read the watchdog timer status and counter.
Serial Communication Interface	1	R_SCI_Set	Configure the SCI pin selection.
	2	R_SCI_Create	SCI channel setup.
	3	R_SCI_Destroy	Shut down a SCI channel.
	4	R_SCI_Send	Send a string of characters.
	5	R_SCI_Receive	Receive a string of characters.
	6	R_SCI_Control	Control the SCI channel.
	7	R_SCI_GetStatus	Check the status of a SCI channel.
CRC calculator	1	R_CRC_Create	Configure the CRC calculator.
	2	R_CRC_Destroy	Shut down the CRC calculator.
	3	R_CRC_Write	Write data into the CRC calculation register.
	4	R_CRC_Read	Read the CRC calculation result.
I <sup>2</sup> C bus interface	1	R_IIC_Create	I <sup>2</sup> C channel setup.
	2	R_IIC_Destroy	Disable an I <sup>2</sup> C channel.
	3	R_IIC_MasterSend	Write data to a slave device.
	4	R_IIC_MasterReceive	Read data from a slave device.
	5	R_IIC_MasterReceiveLast	Complete a DTC-based read process.
	6	R_IIC_SlaveMonitor	Monitor the bus and receive data from a master.
	7	R_IIC_SlaveSend	Write data to a master device.
	8	R_IIC_Control	I <sup>2</sup> C channel control.
	9	R_IIC_GetStatus	Read the status for an I <sup>2</sup> C channel.
Serial Peripheral Interface	1	R_SPI_Create	Configure an SPI channel.
	2	R_SPI_Destroy	Shutdown an SPI channel.
	3	R_SPI_Command	Configure an SPI command.
	4	R_SPI_Transfer	Transfer data over an SPI channel.
	5	R_SPI_Control	Control an SPI channel.
	6	R_SPI_GetStatus	Check the status of an SPI channel.
LIN module	1	R_LIN_Create	Configure the LIN channel.
	2	R_LIN_Destroy	Shutdown the LIN channel.
	3	R_LIN_Transfer	Transfer data on the LIN channel.
	4	R_LIN_Read	Store received data.
	5	R_LIN_Control	Control the LIN channel.
	6	R_LIN_GetStatus	Check the status of the LIN channel.
12-bit Analog to Digital converter	1	R_ADC_12_Set	Select the 12-bit ADC pins.
	2	R_ADC_12_CreateUnit	Configure the 12-bit ADC unit.
	3	R_ADC_12_CreateChannel	Configure the 12-bit ADC unit channels.
	4	R_ADC_12_Destroy	Shut down an ADC unit.
	5	R_ADC_12_Control	Start or stop an ADC unit.
	6	R_ADC_12_Read	Read the ADC conversion results.
10-bit Analog to Digital converter	1	R_ADC_10_Create	Configure an ADC unit.
	2	R_ADC_10_Destroy	Shut down an ADC unit.
	3	R_ADC_10_Control	Start or stop an ADC unit.
	4	R_ADC_10_Read	Read the ADC conversion results.

## 4.2. Description of Each API

This section describes each API and explains how to use them, showing a program example for each. The description of each API is divided into the following items.

<b>Synopsis</b>	Summarises processing by the API function.
<b>Prototype</b>	The function format and a brief explanation of the arguments.
<b>Description</b>	Explains how to use the API function and shows assignable parameters separating each argument with <b>[argument]</b> .
<b>Return value</b>	Describes the returned value of the API function.
<b>Category</b>	Indicates the category of the API function.
<b>Reference</b>	Indicates the API functions to be referred.
<b>Remark</b>	Describes notes to use the API function.
<b>Program example</b>	Represents how to use the API function by a program example. Two examples of return value checking are shown below.

```

/* RPD_L definitions */
#include "r_pdl_pfc.h"
#include "r_pdl_sci.h"

/* RPD_L device-specific definitions */
#include "r_pdl_definitions.h"

void func( void )
{
    bool result;

    /* Write 0xFF to register PFC1 */
    result = (R_PFC_Write(
        1,
        0xFF
    ));
    if (result == false)
    {
        /* Handle the error here */
    }

    /* Keep trying to send a string (if the channel is busy) */
    do
    {
        result = R_SCI_Send(
            2,
            "Renesas RX",
            NULL,
            PDL_NO_FUNC
        );
    } while (result == false);
}

```

For clarity, the return value is not checked in the examples used in this manual.

The RPD\_L API is implemented using function macros. To avoid the possibility of parameters being evaluated more than once do not use operators or function calls within the RPD\_L API parameter list.

## 4.2.1. Clock Generation Circuit

## 1) R\_CGC\_Set

**Synopsis**

Configure the clock generation circuit.

**Prototype**

```
bool R_CGC_Set(
    uint32_t data1, // Input frequency
    uint32_t data2, // System clock frequency
    uint32_t data3, // Peripheral module clock frequency
    uint8_t data4   // Configuration options
);
```

**Description**

Set the clock output frequencies and options.

**[data1]**

The frequency of the main clock oscillator in Hertz.

**[data2]**

The desired frequency of the System clock (ICLK) in Hertz.

**[data3]**

The desired frequency of the Peripheral module clock (PCLK) in Hertz.

**[data4]**Configuration options. The default settings are shown in **bold**.

- Oscillation Stop Detection control

<b>PDL_CGC_OSC_STOP_ENABLE</b> or <b>PDL_CGC_OSC_STOP_DISABLE</b>	Enable or disable the oscillation stop detection function.
--	--

**Return value**

True if all parameters are valid and exclusive; otherwise false.

For RX62G, the following rules shall be checked:

- Main clock oscillator frequency: 8 to 12.5 MHz.
- $f_{\text{ICLK}}$ : 8 to 100 MHz
- $f_{\text{PCLK}}$ : 8 to 50 MHz
- $f_{\text{ICLK}} \geq f_{\text{PCLK}}$
- $f_{\text{ICLK}}$  and  $f_{\text{PCLK}}$  are achievable:  $(\text{main clock oscillator} \times 8) \div 1, 2, 4 \text{ or } 8$

**Category**

Clock generation circuit

**References**

None.

**Remarks**

- This function must be called before configuring clock-dependent modules.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_cgc.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Configure operation using a 12.5 MHz input clock */
    /* ICLK = 100 MHz, PCLK = 50 MHz */
    R_CGC_Set(
        12.5E6,
        100E6,
        50E6,
        PDL_NO_DATA
    );
}
```

## 2) R\_CGC\_GetStatus

### Synopsis

Configure the clock generation circuit.

### Prototype

```
bool R_CGC_GetStatus(
    uint8_t * data    // Pointer to the variable where the status value shall be stored.
);
```

### Description

Read the clock status register.

#### [data]

The status flags shall be stored in the format below.

b7 – b1	b0
0	0: The main clock oscillator is operating normally. 1: The main clock oscillator has stopped.

### Return value

True.

### Category

Clock generation circuit

### References

None.

### Remarks

- None.

### Program example

```
/* RPD_L definitions */
#include "r_pdl_cgc.h"

/* RPD_L device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    uint8_t Status_flags;

    R_CGC_GetStatusSet(
        &Status_flags
    );
}
```

## 4.2.2. Interrupt Control Unit

## 1) R\_INTC\_CreateExtInterrupt

**Synopsis**

Configure an external interrupt pin.

**Prototype**

```
bool R_INTC_CreateExtInterrupt(
    uint8_t data1, // Pin selection
    uint16_t data2, // Configuration
    void * func, // Callback function
    uint8_t data3 // Interrupt priority level
);
```

**Description**

Sets the specified external interrupt.

**[data1]**

Choose the interrupt pin to be configured.

PDL_INTC_IRQn (n = 0 to 7) or PDL_INTC_NMI	IRQn (n = 0 to 7) interrupt pin or NMI interrupt pin
---	---

**[data2]**Choose the pin settings. If multiple selections are required, use “|” to separate each selection. The default settings are shown in **bold**.Options which only apply to the IRQ pins

- Input sense selection

<b>PDL_INTC_LOW</b> or PDL_INTC_FALLING or PDL_INTC_RISING or PDL_INTC_BOTH	Select Low level, Falling edge, Rising edge or Falling and rising edge detection.
--	--

- Alternate pin selection

PDL_INTC_A or PDL_INTC_B or PDL_INTC_C	Select the IRQn-A, IRQn-B or IRQn-C pin to be used (where applicable).
--	--

- DTC trigger control. Not enabled if low-level detection is selected.

<b>PDL_INTC_DTC_TRIGGER_DISABLE</b> or PDL_INTC_DTC_TRIGGER_ENABLE	Disable or enable activation of the DTC when a valid edge transition is detected on an IRQn pin.
---	--

Options which only apply to the NMI

- Input sense selection

<b>PDL_INTC_FALLING</b> or PDL_INTC_RISING	Falling or rising edge detection.
---	-----------------------------------

- Additional detection control

<b>PDL_INTC_LVD_DISABLE</b> or PDL_INTC_LVD_ENABLE	Disable or enable the NMI when a low-voltage detection interrupt occurs.
<b>PDL_INTC_OSD_DISABLE</b> or PDL_INTC_OSD_ENABLE	Disable or enable the NMI when the oscillation stop detection interrupt occurs.

**[func]**

The function to be called when a valid condition is detected.

Specify PDL\_NO\_FUNC if no IRQn interrupt is required.

A function must be specified for the NMI pin.

**[data3]**

The IRQn interrupt priority level. Select between 1 (lowest priority) and 15 (highest priority).

This parameter will be ignored if PDL\_NO\_FUNC is specified for parameter func.

This value does not apply to the NMI pin and is ignored.

**Return value**

True if all parameters are valid and exclusive; otherwise false.

<b>Category</b>	Interrupt control
<b>Reference</b>	R_INTC_ControlExtInterrupt, R_INTC_GetExtInterruptStatus
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• The selected interrupt pin is enabled automatically.</li> <li>• Port Function Control registers PF8IRQ or PF9IRQ are modified to select the IRQn pin. For the smaller device packages, some B and C pin options are not available. For the 64-pin package, IRQ2, IRQ4, IRQ6 and IRQ7 are not available.</li> <li>• The appropriate I/O port ICR and DDR registers are modified.</li> <li>• Please check that the selected IRQ pin is available on the selected device package.</li> <li>• Please see the notes on callback function use in §6.</li> <li>• The NMI callback function should not return. It should stop operation or reset the system.</li> <li>• If the NMI interrupt fails to initialise, this function will return false.</li> </ul>

**Program example**

```

/* RPD_L definitions */
#include "r_pdl_intc.h"

/* RPD_L device-specific definitions */
#include "r_pdl_definitions.h"

/* Declaration of callback function */
void CallBackFunc( void );

void func( void )
{
    /* Configure the IRQ0 interrupt on pin IRQ0-A */
    R_INTC_CreateExtInterrupt(
        PDL_INTC_IRQ0,
        PDL_INTC_FALLING | PDL_INTC_A,
        CallBackFunc,
        7
    );
}

```



## 2) R\_INTC\_CreateSoftwareInterrupt

### Synopsis

Enable use of the software interrupt.

### Prototype

```
bool R_INTC_CreateSoftwareInterrupt(
    uint8_t data1, // Configuration
    void * func,   // Callback function
    uint8_t data2  // Interrupt priority level
);
```

### Description

Configure and enable the software interrupt.

#### [data1]

Choose the pin settings. The default setting is shown in **bold**.

- DTC trigger control.

<b>PDL_INTC_DTC_SW_TRIGGER_DISABLE</b> or PDL_INTC_DTC_SW_TRIGGER_ENABLE	Disable or enable activation of the DTC when a software interrupt is generated.
---	---

#### [func]

The function to be called when a valid condition is detected.  
Specify PDL\_NO\_FUNC if no interrupt is required.

#### [data2]

The interrupt priority level. Select between 1 (lowest priority) and 15 (highest priority).  
This parameter will be ignored if PDL\_NO\_FUNC is specified for parameter func.

### Return value

True if all parameters are valid; otherwise false.

### Category

Interrupt control

### Reference

R\_INTC\_Write

### Remarks

- Please see the notes on callback function use in §6.
- Specifying PDL\_NO\_FUNC for the callback function allows the software interrupt to be used as a DTC trigger.
- Use R\_INTC\_Write to generate the software interrupt.

### Program example

```
/* RPDL definitions */
#include "r_pdl_intc.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

/* Declaration of callback function */
void CallBackFunc( void );

void func( void )
{
    /* Configure the software interrupt handler */
    R_INTC_CreateSoftwareInterrupt(
        PDL_NO_DATA,
        CallBackFunc,
        7
    );
}
```

### 3) R\_INTC\_CreateFastInterrupt

#### Synopsis

Enable faster interrupt processing for one interrupt.

#### Prototype

```
bool R_INTC_CreateFastInterrupt(
    uint8_t data    // The interrupt to be selected
);
```

#### Description (1/2)

#### [data]

Choose the interrupt vector to be processed using the fast interrupt process.

Name	Module	Interrupt cause
PDL_INTC_VECTOR_BUSERR	External bus	Error (illegal access or timeout)
PDL_INTC_VECTOR_FIFERR	Flash memory	Error
PDL_INTC_VECTOR_FRDYI		Ready
PDL_INTC_VECTOR_SWINT	Interrupt control	Software interrupt
PDL_INTC_VECTOR_CMT0	Compare match timer	Compare match
PDL_INTC_VECTOR_CMT1		
PDL_INTC_VECTOR_CMT2		
PDL_INTC_VECTOR_CMT3		
PDL_INTC_VECTOR_SPEI0	RSPI channel 0	Error
PDL_INTC_VECTOR_SPRI0		Receive buffer full
PDL_INTC_VECTOR_SPTI0		Transmit buffer empty
PDL_INTC_VECTOR_SPII0		Idle
PDL_INTC_VECTOR_ERS0	CAN channel 0	Error
PDL_INTC_VECTOR_RXF0		Receive FIFO
PDL_INTC_VECTOR_TXF0		Transmit FIFO
PDL_INTC_VECTOR_RXM0		Reception complete
PDL_INTC_VECTOR_TXM0		Transmission complete
PDL_INTC_VECTOR_IRQ0	External interrupt pin	Valid edge or level detected
PDL_INTC_VECTOR_IRQ1		
PDL_INTC_VECTOR_IRQ2		
PDL_INTC_VECTOR_IRQ3		
PDL_INTC_VECTOR_IRQ4		
PDL_INTC_VECTOR_IRQ5		
PDL_INTC_VECTOR_IRQ6		
PDL_INTC_VECTOR_IRQ7		
PDL_INTC_VECTOR_WOVI	Watchdog timer	Overflow
PDL_INTC_VECTOR_ADIO	10-bit ADC	Conversion completed
PDL_INTC_VECTOR_S12ADI0	12-bit ADC	Conversion completed
PDL_INTC_VECTOR_S12ADI1		
PDL_INTC_VECTOR_CMPI	Multi-function Timer Pulse Unit, channel 0	Comparison condition detected
PDL_INTC_VECTOR_TGIA0		Compare match or Input capture A
PDL_INTC_VECTOR_TGIB0		Compare match or Input capture B
PDL_INTC_VECTOR_TGIC0		Compare match or Input capture C
PDL_INTC_VECTOR_TGID0		Compare match or Input capture D
PDL_INTC_VECTOR_TCIV0		Overflow
PDL_INTC_VECTOR_TGIE0		Compare match E
PDL_INTC_VECTOR_TGIF0		Compare match F
PDL_INTC_VECTOR_TGIA1	Multi-function Timer Pulse Unit, channel 1	Compare match or Input capture A
PDL_INTC_VECTOR_TGIB1		Compare match or Input capture B
PDL_INTC_VECTOR_TCIV1		Overflow
PDL_INTC_VECTOR_TCIU1		Underflow
PDL_INTC_VECTOR_TGIA2	Multi-function Timer Pulse Unit, channel 2	Compare match or Input capture A
PDL_INTC_VECTOR_TGIB2		Compare match or Input capture B
PDL_INTC_VECTOR_TCIV2		Overflow
PDL_INTC_VECTOR_TCIU2		Underflow
PDL_INTC_VECTOR_TGIA3	Multi-function Timer Pulse Unit, channel 3	Compare match or Input capture A
PDL_INTC_VECTOR_TGIB3		Compare match or Input capture B
PDL_INTC_VECTOR_TGIC3		Compare match or Input capture C
PDL_INTC_VECTOR_TGID3		Compare match or Input capture D
PDL_INTC_VECTOR_TCIV3		Overflow

Description (2/2)		
PDL_INTC_VECTOR_TGIA4	Multi-function Timer Pulse Unit, channel 4	Compare match or Input capture A
PDL_INTC_VECTOR_TGIB4		Compare match or Input capture B
PDL_INTC_VECTOR_TGIC4		Compare match or Input capture C
PDL_INTC_VECTOR_TGID4		Compare match or Input capture D
PDL_INTC_VECTOR_TCIV4		Overflow
PDL_INTC_VECTOR_TGIU5	Multi-function Timer Pulse Unit, channel 5	Compare match or Input capture U
PDL_INTC_VECTOR_TGIV5		Compare match or Input capture V
PDL_INTC_VECTOR_TGIW5		Compare match or Input capture W
PDL_INTC_VECTOR_TGIA6		Compare match or Input capture A
PDL_INTC_VECTOR_TGIB6		Compare match or Input capture B
PDL_INTC_VECTOR_TGIC6	Multi-function Timer Pulse Unit, channel 6	Compare match or Input capture C
PDL_INTC_VECTOR_TGID6		Compare match or Input capture D
PDL_INTC_VECTOR_TCIV6		Overflow
PDL_INTC_VECTOR_TGIA7		Compare match or Input capture A
PDL_INTC_VECTOR_TGIB7		Compare match or Input capture B
PDL_INTC_VECTOR_TGIC7	Multi-function Timer Pulse Unit, channel 7	Compare match or Input capture C
PDL_INTC_VECTOR_TGID7		Compare match or Input capture D
PDL_INTC_VECTOR_TCIV7		Overflow
PDL_INTC_VECTOR_OEI1		
PDL_INTC_VECTOR_OEI2		
PDL_INTC_VECTOR_OEI3	Port Output Enable	Input-level sampling or output-level comparison detection
PDL_INTC_VECTOR_OEI4		
PDL_INTC_VECTOR_GTCIA0		
PDL_INTC_VECTOR_GTCIB0		
PDL_INTC_VECTOR_GTCIC0		
PDL_INTC_VECTOR_GTCIE0	General PWM timer, channel 0	Compare match or input capture A
PDL_INTC_VECTOR_GTCIV0		Compare match or input capture B
PDL_INTC_VECTOR_LOCO1		Compare match C or D
PDL_INTC_VECTOR_GTCIA1		Compare match E or F
PDL_INTC_VECTOR_GTCIB1		Counter limit match
PDL_INTC_VECTOR_GTCIC1	General PWM timer, channel 1	LOCO count function event
PDL_INTC_VECTOR_GTCIE1		Compare match or input capture A
PDL_INTC_VECTOR_GTCIV1		Compare match or input capture B
PDL_INTC_VECTOR_GTCIA2		Compare match C or D
PDL_INTC_VECTOR_GTCIB2		Compare match E or F
PDL_INTC_VECTOR_GTCIC2	General PWM timer, channel 2	Counter limit match
PDL_INTC_VECTOR_GTCIE2		Compare match or input capture A
PDL_INTC_VECTOR_GTCIV2		Compare match or input capture B
PDL_INTC_VECTOR_GTCIA3		Compare match C or D
PDL_INTC_VECTOR_GTCIB3		Compare match E or F
PDL_INTC_VECTOR_GTCIC3	General PWM timer, channel 3	Counter limit match
PDL_INTC_VECTOR_GTCIE3		Compare match or input capture A
PDL_INTC_VECTOR_GTCIV3		Compare match or input capture B
PDL_INTC_VECTOR_ERI0		Compare match C or D
PDL_INTC_VECTOR_RXI0		Compare match E or F
PDL_INTC_VECTOR_TXI0	SCI, channel 0	Counter limit match
PDL_INTC_VECTOR_TEI0		Compare match or input capture A
PDL_INTC_VECTOR_ERI1		Compare match or input capture B
PDL_INTC_VECTOR_RXI1		Compare match C or D
PDL_INTC_VECTOR_TXI1		Compare match E or F
PDL_INTC_VECTOR_TEI1	SCI, channel 1	Counter limit match
PDL_INTC_VECTOR_ERI2		Compare match or input capture A
PDL_INTC_VECTOR_RXI2		Compare match or input capture B
PDL_INTC_VECTOR_TXI2		Compare match C or D
PDL_INTC_VECTOR_TEI2		Compare match E or F
PDL_INTC_VECTOR_ICEEI0	I <sup>2</sup> C bus interface, channel 0	Counter limit match
PDL_INTC_VECTOR_ICRXI0		Compare match or input capture A
PDL_INTC_VECTOR_ICTXI0		Compare match or input capture B
PDL_INTC_VECTOR_ICTEI0		Compare match C or D
PDL_INTC_VECTOR_LIN0		Compare match E or F
	LIN channel 0	Counter limit match
		Transfer error or event generation
		Data received
		Start of next data transfer
		End of data transfer

Return value

True.

Category

Interrupt control

**Reference****Remarks**

- The fast interrupt processing is allocated to only one interrupt handler.
- Open the file `r_pdl_user_definitions.h` and edit the definition `FAST_INTC_VECTOR` to give it the same value as the interrupt vector used in parameter `data1`.  
For example:  

```
#define FAST_INTC_VECTOR  PDL_INTC_VECTOR_ADIO
```

 This will direct the compiler to generate the instructions required for a fast interrupt vector.
- This function uses an interrupt routine to modify the `FINTV` register. If the user has disabled interrupts (cleared the 'I' bit in the `PSW` register) in their own code, this function will lock up.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_intc.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func( void )
{
    /* Assign the fast interrupt to the handler for pin IRQ3 */
    R_INTC_CreateFastInterrupt(
        PDL_INTC_VECTOR_IRQ3
    );
}

/* Remember to edit r_pdl_user_definitions.h (see remark 2) */
```

#### 4) R\_INTC\_CreateExceptionHandlers

##### Synopsis

Assign handlers for the fixed-vector interrupts.

##### Prototype

```
bool R_INTC_CreateExceptionHandlers(
    void *func1,    // Callback function
    void *func2,    // Callback function
    void *func3     // Callback function
);
```

##### Description

Register the user functions to be called by the fixed-vector and software interrupts.

##### [func1]

The function to be called when a privileged instruction is detected while in user mode. Specify PDL\_NO\_FUNC if no callback function is required.

##### [func2]

The function to be called when an undefined instruction is detected. Specify PDL\_NO\_FUNC if no callback function is required.

##### [func3]

The function to be called when a floating point exception is detected. Specify PDL\_NO\_FUNC if no callback function is required.

##### Return value

True.

##### Category

Interrupt control

##### Reference

##### Remarks

- Please see the notes on callback function use in §6.
- A callback function is executed by the interrupt processing function. This means that no other interrupt can be processed until the callback function has completed.
- If access exceptions are to be handled, in file Interrupt\_INT.c modify the second entry in the Fixed\_Vectors array.

##### Program example

```
/* RPDL definitions */
#include "r_pdl_intc.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

/* Declaration of callback function */
void CallBackFunc( void );

void func( void )
{
    /* Add a function to manage floating point errors */
    R_INTC_CreateExceptionHandlers(
        PDL_NO_FUNC,
        PDL_NO_FUNC,
        FloatingPointFunc
    );
}
```

## 5) R\_INTC\_ControlExtInterrupt

### Synopsis

External interrupt control.

### Prototype

```
bool R_INTC_ControlExtInterrupt(
    uint8_t data1,    // Pin selection
    uint16_t data2    // Control
);
```

### Description

Modifies the specified external interrupt.

#### [data1]

Choose the interrupt pin to be controlled.

PDL_INTC_IRQn (n = 0 to 7) or PDL_INTC_NMI	IRQn interrupt pin or NMI interrupt pin
---	--

#### [data2]

Select the controls. If multiple selections are required, use "[" to separate each selection.

- Enable or disable the interrupt pin (for the IRQ pins)

PDL_INTC_ENABLE or PDL_INTC_DISABLE	Enable or disable the IRQn interrupt pin.
--	---

- Detection sense selection (for the IRQ pins)

PDL_INTC_LOW or	Low level detection
PDL_INTC_FALLING or	Falling edge detection
PDL_INTC_RISING or	Rising edge detection
PDL_INTC_BOTH	Falling and rising edge detection

- Interrupt request clearing

PDL_INTC_CLEAR_IR_FLAG	<p>Clear the Interrupt Request flag. This is not required if:</p> <ul style="list-style-type: none"> <li>• A callback function has been specified.</li> <li>• The interrupt priority level is higher than 0.</li> <li>• The processor interrupt priority level is lower than the interrupt priority level.</li> </ul> <p>This operation will not work for a level-sensitive interrupt if the input signal is still low.</p>
PDL_INTC_CLEAR_OSD_FLAG	Clear the Oscillation Stop detection flag.

### Return value

True if all parameters are valid and exclusive; otherwise false.

### Category

Interrupt control

### Reference

R\_INTC\_CreateExtInterrupt, R\_INTC\_GetExtInterruptStatus

### Remarks

- The NMI pin was enabled during R\_INTC\_CreateExtInterrupt and cannot be disabled (an MCU design feature).
- When disabling an IRQn pin, the Interrupt Request flag will be cleared automatically.
- A callback function may be called once more if a valid event occurs just before the interrupt pin is disabled.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_intc.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func( void )
{
    /* Disable the IRQ1 interrupt pin and clear the flag */
    R_INTC_ControlExtInterrupt(
        PDL_INTC_IRQ1,
        PDL_INTC_DISABLE | PDL_INTC_CLEAR_IR_FLAG
    );
}
```

## 6) R\_INTC\_GetExtInterruptStatus

### Synopsis

Read the external interrupt status.

### Prototype

```
bool R_INTC_GetExtInterruptStatus(
    uint8_t data1,    // Pin selection
    uint8_t * data2   // A pointer to the buffer where the status data shall be stored.
);
```

### Description

Acquire the status for the specified external interrupt.

#### [data1]

Choose the interrupt pin to be checked.

PDL_INTC_IRQn (n = 0 to 7) or PDL_INTC_NMI	IRQn (n = 0 to 7) interrupt pin or NMI interrupt pin
---	---

#### [data2]

The status flags shall be stored in the following format:

For an IRQ pin:

b7 – b4	b3 – b2	b1	b0
	Input detection condition	Pin level	Input detection status
0	00b: Low level 01b: Falling edge 10b: Rising edge 11b: Both edges	0: Low 1: High	0: Not detected 1: Detected

For the NMI pin:

b7 – b4	b3	b2	b1	b0
	Oscillation stop interrupt request	Low voltage interrupt request	Input detection condition	Input detection status
0	0: Not detected 1: Detected	0: Not detected 1: Detected	0: Falling edge 1: Rising edge	0: Not detected 1: Detected

### Return value

True if all parameters are valid and exclusive; otherwise false.

### Category

Interrupt control

### Reference

R\_INTC\_CreateExtInterrupt, R\_INTC\_ControlExtInterrupt

### Remarks

- Port Function Control registers PF8IRQ or PF9IRQ may be checked to determine which pin is used for IRQn.
- If this function is called from within a callback function, the input detection status will be 0.

### Program example

```
/* RPD_L definitions */
#include "r_pdl_intc.h"

/* RPD_L device-specific definitions */
#include "r_pdl_definitions.h"

void func( void )
{
    uint8_t irq_status;

    /* Read the IR flag and pin state for IRQ5 */
    R_INTC_GetExtInterruptStatus(
        PDL_INTC_IRQ5,
        &irq_status
    );
}
```



The INTC Read, Write and Modify functions use one of the following register definitions.

IR register definitions

PDL_INTC_REG_IR_BSC_BUSERR	PDL_INTC_REG_IR_MTU5_TGIU
PDL_INTC_REG_IR_FCUIF_FIFER	PDL_INTC_REG_IR_MTU5_TGIV
PDL_INTC_REG_IR_FCUIF_FRDYI	PDL_INTC_REG_IR_MTU5_TGIW
PDL_INTC_REG_IR_ICU_SWINT	PDL_INTC_REG_IR_MTU6_TGIA
PDL_INTC_REG_IR_CMT0_CMI	PDL_INTC_REG_IR_MTU6_TGIB
PDL_INTC_REG_IR_CMT1_CMI	PDL_INTC_REG_IR_MTU6_TGIC
PDL_INTC_REG_IR_CMT2_CMI	PDL_INTC_REG_IR_MTU6_TGID
PDL_INTC_REG_IR_CMT3_CMI	PDL_INTC_REG_IR_MTU6_TCIV
PDL_INTC_REG_IR_SPI0_SPEI	PDL_INTC_REG_IR_MTU7_TGIA
PDL_INTC_REG_IR_SPI0_SPRI	PDL_INTC_REG_IR_MTU7_TGIB
PDL_INTC_REG_IR_SPI0_SPTI	PDL_INTC_REG_IR_MTU7_TGIC
PDL_INTC_REG_IR_SPI0_SPII	PDL_INTC_REG_IR_MTU7_TGID
PDL_INTC_REG_IR_CAN0_ERS	PDL_INTC_REG_IR_MTU7_TCIV
PDL_INTC_REG_IR_CAN0_RXF	PDL_INTC_REG_IR_POE_OEI1
PDL_INTC_REG_IR_CAN0_TXF	PDL_INTC_REG_IR_POE_OEI2
PDL_INTC_REG_IR_CAN0_RXM	PDL_INTC_REG_IR_POE_OEI3
PDL_INTC_REG_IR_CAN0_TXM	PDL_INTC_REG_IR_POE_OEI4
PDL_INTC_REG_IR_ICU_IRQ0	PDL_INTC_REG_IR_GPT0_GTCIA
PDL_INTC_REG_IR_ICU_IRQ1	PDL_INTC_REG_IR_GPT0_GTCIB
PDL_INTC_REG_IR_ICU_IRQ2	PDL_INTC_REG_IR_GPT0_GTCIC
PDL_INTC_REG_IR_ICU_IRQ3	PDL_INTC_REG_IR_GPT0_GTCIE
PDL_INTC_REG_IR_ICU_IRQ4	PDL_INTC_REG_IR_GPT0_GTCIV
PDL_INTC_REG_IR_ICU_IRQ5	PDL_INTC_REG_IR_GPT0_LOCO1
PDL_INTC_REG_IR_ICU_IRQ6	PDL_INTC_REG_IR_GPT1_GTCIA
PDL_INTC_REG_IR_ICU_IRQ7	PDL_INTC_REG_IR_GPT1_GTCIB
PDL_INTC_REG_IR_WDT_WOVI	PDL_INTC_REG_IR_GPT1_GTCIC
PDL_INTC_REG_IR_AD0_ADI	PDL_INTC_REG_IR_GPT1_GTCIE
PDL_INTC_REG_IR_S12ADA0_ADI	PDL_INTC_REG_IR_GPT1_GTCIV
PDL_INTC_REG_IR_S12ADA1_ADI	PDL_INTC_REG_IR_GPT2_GTCIA
PDL_INTC_REG_IR_CMPI	PDL_INTC_REG_IR_GPT2_GTCIB
PDL_INTC_REG_IR_MTU0_TGIA	PDL_INTC_REG_IR_GPT2_GTCIC
PDL_INTC_REG_IR_MTU0_TGIB	PDL_INTC_REG_IR_GPT2_GTCIE
PDL_INTC_REG_IR_MTU0_TGIC	PDL_INTC_REG_IR_GPT2_GTCIV
PDL_INTC_REG_IR_MTU0_TGID	PDL_INTC_REG_IR_GPT3_GTCIA
PDL_INTC_REG_IR_MTU0_TCIV	PDL_INTC_REG_IR_GPT3_GTCIB
PDL_INTC_REG_IR_MTU0_TGIE	PDL_INTC_REG_IR_GPT3_GTCIC
PDL_INTC_REG_IR_MTU0_TGIF	PDL_INTC_REG_IR_GPT3_GTCIE
PDL_INTC_REG_IR_MTU1_TGIA	PDL_INTC_REG_IR_GPT3_GTCIV
PDL_INTC_REG_IR_MTU1_TGIB	PDL_INTC_REG_IR_SCI0_ERI
PDL_INTC_REG_IR_MTU1_TCIV	PDL_INTC_REG_IR_SCI0_RXI
PDL_INTC_REG_IR_MTU1_TCIU	PDL_INTC_REG_IR_SCI0_TXI
PDL_INTC_REG_IR_MTU2_TGIA	PDL_INTC_REG_IR_SCI0_TEI
PDL_INTC_REG_IR_MTU2_TGIB	PDL_INTC_REG_IR_SCI1_ERI
PDL_INTC_REG_IR_MTU2_TCIV	PDL_INTC_REG_IR_SCI1_RXI
PDL_INTC_REG_IR_MTU2_TCIU	PDL_INTC_REG_IR_SCI1_TXI
PDL_INTC_REG_IR_MTU3_TGIA	PDL_INTC_REG_IR_SCI1_TEI
PDL_INTC_REG_IR_MTU3_TGIB	PDL_INTC_REG_IR_SCI2_ERI
PDL_INTC_REG_IR_MTU3_TGIC	PDL_INTC_REG_IR_SCI2_RXI
PDL_INTC_REG_IR_MTU3_TGID	PDL_INTC_REG_IR_SCI2_TXI
PDL_INTC_REG_IR_MTU3_TCIV	PDL_INTC_REG_IR_SCI2_TEI
PDL_INTC_REG_IR_MTU4_TGIA	PDL_INTC_REG_IR_IIC0_EEI
PDL_INTC_REG_IR_MTU4_TGIB	PDL_INTC_REG_IR_IIC0_RXI
PDL_INTC_REG_IR_MTU4_TGIC	PDL_INTC_REG_IR_IIC0_TXI
PDL_INTC_REG_IR_MTU4_TGID	PDL_INTC_REG_IR_IIC0_TEI
PDL_INTC_REG_IR_MTU4_TCIV	PDL_INTC_REG_IR_LIN0_LIN

## IER register definitions

PDL_INTC_REG_IER02	PDL_INTC_REG_IER12
PDL_INTC_REG_IER03	PDL_INTC_REG_IER13
PDL_INTC_REG_IER05	PDL_INTC_REG_IER15
PDL_INTC_REG_IER07	PDL_INTC_REG_IER16
PDL_INTC_REG_IER08	PDL_INTC_REG_IER17
PDL_INTC_REG_IER0C	PDL_INTC_REG_IER18
PDL_INTC_REG_IER0D	PDL_INTC_REG_IER1A
PDL_INTC_REG_IER0E	PDL_INTC_REG_IER1B
PDL_INTC_REG_IER0F	PDL_INTC_REG_IER1C
PDL_INTC_REG_IER10	PDL_INTC_REG_IER1D
PDL_INTC_REG_IER11	PDL_INTC_REG_IER1E
	PDL_INTC_REG_IER1F

## IPR register definitions

PDL_INTC_REG_IPR_BSC_BUSERR	PDL_INTC_REG_IPR_MTU3_TGIA
PDL_INTC_REG_IPR_FCUIF_FIFERR	PDL_INTC_REG_IPR_MTU3_TGIB
PDL_INTC_REG_IPR_FCUIF_FRDYI	PDL_INTC_REG_IPR_MTU3_TGIC
PDL_INTC_REG_IPR_ICU_SWINT	PDL_INTC_REG_IPR_MTU3_TGID
PDL_INTC_REG_IPR_CMT0_CMI	PDL_INTC_REG_IPR_MTU3_TCIV
PDL_INTC_REG_IPR_CMT1_CMI	PDL_INTC_REG_IPR_MTU4_TGIA
PDL_INTC_REG_IPR_CMT2_CMI	PDL_INTC_REG_IPR_MTU4_TGIB
PDL_INTC_REG_IPR_CMT3_CMI	PDL_INTC_REG_IPR_MTU4_TGIC
PDL_INTC_REG_IPR_SPI0_SPEI	PDL_INTC_REG_IPR_MTU4_TGID
PDL_INTC_REG_IPR_SPI0_SPRI	PDL_INTC_REG_IPR_MTU4_TCIV
PDL_INTC_REG_IPR_SPI0_SPTI	PDL_INTC_REG_IPR_MTU5_TGIU
PDL_INTC_REG_IPR_SPI0_SPII	PDL_INTC_REG_IPR_MTU5_TGIV
PDL_INTC_REG_IPR_CAN0_ERS	PDL_INTC_REG_IPR_MTU5_TGIW
PDL_INTC_REG_IPR_CAN0_RXF	PDL_INTC_REG_IPR_MTU6_TGIA
PDL_INTC_REG_IPR_CAN0_TXF	PDL_INTC_REG_IPR_MTU6_TGIB
PDL_INTC_REG_IPR_CAN0_RXM	PDL_INTC_REG_IPR_MTU6_TGIC
PDL_INTC_REG_IPR_CAN0_TXM	PDL_INTC_REG_IPR_MTU6_TGID
PDL_INTC_REG_IPR_ICU_IRQ0	PDL_INTC_REG_IPR_MTU6_TCIV
PDL_INTC_REG_IPR_ICU_IRQ1	PDL_INTC_REG_IPR_MTU7_TGIA
PDL_INTC_REG_IPR_ICU_IRQ2	PDL_INTC_REG_IPR_MTU7_TGIB
PDL_INTC_REG_IPR_ICU_IRQ3	PDL_INTC_REG_IPR_MTU7_TGIC
PDL_INTC_REG_IPR_ICU_IRQ4	PDL_INTC_REG_IPR_MTU7_TGID
PDL_INTC_REG_IPR_ICU_IRQ5	PDL_INTC_REG_IPR_MTU7_TCIV
PDL_INTC_REG_IPR_ICU_IRQ6	PDL_INTC_REG_IPR_POE_OEI1
PDL_INTC_REG_IPR_ICU_IRQ7	PDL_INTC_REG_IPR_POE_OEI2
PDL_INTC_REG_IPR_WDT_WOVI	PDL_INTC_REG_IPR_POE_OEI3
PDL_INTC_REG_IPR_AD0_ADI	PDL_INTC_REG_IPR_POE_OEI4
PDL_INTC_REG_IPR_S12ADA0_ADI	PDL_INTC_REG_IPR_GPT0_GTCIA
PDL_INTC_REG_IPR_S12ADA1_ADI	PDL_INTC_REG_IPR_GPT0_GTCIB
PDL_INTC_REG_IPR_CMPI	PDL_INTC_REG_IPR_GPT0_GTCIC
PDL_INTC_REG_IPR_MTU0_TGIA	PDL_INTC_REG_IPR_GPT0_GTCIE
PDL_INTC_REG_IPR_MTU0_TGIB	PDL_INTC_REG_IPR_GPT0_GTCIV
PDL_INTC_REG_IPR_MTU0_TGIC	PDL_INTC_REG_IPR_GPT0_LOCO1
PDL_INTC_REG_IPR_MTU0_TGID	PDL_INTC_REG_IPR_GPT1_GTCIA
PDL_INTC_REG_IPR_MTU0_TCIV	PDL_INTC_REG_IPR_GPT1_GTCIB
PDL_INTC_REG_IPR_MTU0_TGIE	PDL_INTC_REG_IPR_GPT1_GTCIC
PDL_INTC_REG_IPR_MTU0_TGIF	PDL_INTC_REG_IPR_GPT1_GTCIE
PDL_INTC_REG_IPR_MTU1_TGIA	PDL_INTC_REG_IPR_GPT1_GTCIV
PDL_INTC_REG_IPR_MTU1_TGIB	PDL_INTC_REG_IPR_GPT2_GTCIA
PDL_INTC_REG_IPR_MTU1_TCIV	PDL_INTC_REG_IPR_GPT2_GTCIB
PDL_INTC_REG_IPR_MTU1_TCIU	PDL_INTC_REG_IPR_GPT2_GTCIC
PDL_INTC_REG_IPR_MTU2_TGIA	PDL_INTC_REG_IPR_GPT2_GTCIE
PDL_INTC_REG_IPR_MTU2_TGIB	PDL_INTC_REG_IPR_GPT2_GTCIV
PDL_INTC_REG_IPR_MTU2_TCIV	PDL_INTC_REG_IPR_GPT3_GTCIA
PDL_INTC_REG_IPR_MTU2_TCIU	PDL_INTC_REG_IPR_GPT3_GTCIB
	PDL_INTC_REG_IPR_GPT3_GTCIC
	PDL_INTC_REG_IPR_GPT3_GTCIE
	PDL_INTC_REG_IPR_GPT3_GTCIV

---

PDL_INTC_REG_IPR_SCI0_ERI	PDL_INTC_REG_IPR_SCI2_ERI
PDL_INTC_REG_IPR_SCI0_RXI	PDL_INTC_REG_IPR_SCI2_RXI
PDL_INTC_REG_IPR_SCI0_TXI	PDL_INTC_REG_IPR_SCI2_TXI
PDL_INTC_REG_IPR_SCI0_TEI	PDL_INTC_REG_IPR_IIC0_ICEEI
PDL_INTC_REG_IPR_SCI1_ERI	PDL_INTC_REG_IPR_IIC0_ICRXI
PDL_INTC_REG_IPR_SCI1_RXI	PDL_INTC_REG_IPR_IIC0_ICTXI
PDL_INTC_REG_IPR_SCI1_TXI	PDL_INTC_REG_IPR_IIC0 ICTEI
PDL_INTC_REG_IPR_SCI1_TEI	PDL_INTC_REG_IPR_LIN0_LIN

## 7) R\_INTC\_Read

### Synopsis

Read an interrupt register.

### Prototype

```
bool R_INTC_Read(
    uint16_t data1, // Register selection
    uint8_t * data2 // Data storage location
);
```

### Description

Read an interrupt register and store the value.

#### [data1]

- The register to be read.

PDL_INTC_REG_IPL or PDL_INTC_REG_IR_(register) or PDL_INTC_REG_IER_(register) or PDL_INTC_REG_IPR_(register) or PDL_INTC_REG_DTCER_(register)	Select the current CPU interrupt priority level or Interrupt Request register or Interrupt Request Enable register or Interrupt Priority register or DTC Activation Enable register
---	---

#### [data2]

The location where the register's value shall be stored.

### Return value

True if all parameters are valid and exclusive; otherwise false.

### Category

Interrupt control

### Reference

None.

### Remarks

- For (register), select one of the registers listed in the tables starting on page 57.

### Program example

```
/* RPDL definitions */
#include "r_pdl_intc.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func( void )
{
    uint8_t ipl;

    /* Read the IPL bits */
    R_INTC_Read(
        PDL_INTC_REG_IPL,
        &ipl
    );
}
```

## 8) R\_INTC\_Write

**Synopsis**

Update an interrupt register.

**Prototype**

```
bool R_INTC_Write(
    uint16_t data1, // Register selection
    uint8_t data2   // Register value
);
```

**Description**

Write the new value to an interrupt register.

**[data1]**

- The register to be updated.

PDL\_INTC\_REG\_IPL or  
PDL\_INTC\_REG\_IR\_(register) or  
PDL\_INTC\_REG\_IER\_(register) or  
PDL\_INTC\_REG\_IPR\_(register) or  
PDL\_INTC\_REG\_DTCEr\_(register) or  
PDL\_INTC\_REG\_SWINTR

Select the current CPU interrupt priority level or  
Interrupt Request register or  
Interrupt Request Enable register or  
Interrupt Priority register or  
DTC Activation Enable register or  
Software interrupt activation register

**[data2]**

The value to be written to the register.

**Return value**

True if the parameter is within range; otherwise false.

**Category**

Interrupt control

**Reference**

None.

**Remarks**

- This function uses an interrupt routine to modify the IPL bits. If the user has disabled interrupts (cleared the 'I' bit in the PSW register) in their own code, this function will lock up.
- For (register), select one of the registers listed in the tables starting on page 57.
- Write 1 to the SWINTR register to generate a software interrupt request.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_intc.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func( void )
{
    /* Set the IPL to 6 */
    R_INTC_Write(
        PDL_INTC_REG_IPL,
        6
    );

    /* Set the IR for IRQ0 to 0 */
    R_INTC_Write(
        PDL_INTC_REG_IR_ICU_IRQ0,
        0
    );
}
```

## 9) R\_INTC\_Modify

### Synopsis

Modify an interrupt register.

### Prototype

```
bool R_INTC_Modify(
    uint16_t data1, // Register selection
    uint8_t data2,  // Logical operation
    uint8_t data3   // Modification value
);
```

### Description

Update the value in an interrupt register.

#### [data1]

- The register to be updated.

PDL_INTC_REG_IR_(register) or PDL_INTC_REG_IER_(register) or PDL_INTC_REG_IPR_(register)	Select the Interrupt Request register or Interrupt Request Enable register or Interrupt Priority register
--	---

#### [data2]

- The logical operation to be applied to the register contents.

PDL_INTC_AND or PDL_INTC_OR or PDL_INTC_XOR	Select between AND (&), OR ( ) or Exclusive-OR (^).
---	---

#### [data3]

The value to be used by the logical operation.

### Return value

True if the parameter is within range; otherwise false.

### Category

Interrupt control

### Reference

None.

### Remarks

- This function uses an interrupt routine to modify the IPL bits. If the user has disabled interrupts (cleared the 'I' bit in the PSW register) in their own code, this function will lock up.
- For (register), select one of the registers listed in the tables starting on page 57.

### Program example

```
/* RPDL definitions */
#include "r_pdl_intc.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func( void )
{
    /* Set bits 6 and 4 in IER09 to 1 */
    R_INTC_Modify(
        PDL_INTC_REG_IER09,
        PDL_INTC_OR,
        0x50
    );
}
```

## 4.2.3. I/O Port

Each I/O Port function can operate on a complete port, or on individual port pins. The definitions available to functions are listed below.

## I/O port definitions

PDL_IO_PORT_1	Port P1	PDL_IO_PORT_8	Port P8
PDL_IO_PORT_2	Port P2	PDL_IO_PORT_9	Port P9
PDL_IO_PORT_3	Port P3	PDL_IO_PORT_A	Port PA
PDL_IO_PORT_4	Port P4	PDL_IO_PORT_B	Port PB
PDL_IO_PORT_5	Port P5	PDL_IO_PORT_D	Port PD
PDL_IO_PORT_6	Port P6	PDL_IO_PORT_E	Port PE
PDL_IO_PORT_7	Port P7	PDL_IO_PORT_G	Port PG

Note: Refer to the hardware manual for the ports which are available on the device that you have selected.

## I/O port pin definitions

PDL_IO_PORT_1_0	Port pin P1 <sub>0</sub>	PDL_IO_PORT_6_0	Port pin P6 <sub>0</sub>	PDL_IO_PORT_B_0	Port pin PB <sub>0</sub>
PDL_IO_PORT_1_1	Port pin P1 <sub>1</sub>	PDL_IO_PORT_6_1	Port pin P6 <sub>1</sub>	PDL_IO_PORT_B_1	Port pin PB <sub>1</sub>
		PDL_IO_PORT_6_2	Port pin P6 <sub>2</sub>	PDL_IO_PORT_B_2	Port pin PB <sub>2</sub>
PDL_IO_PORT_2_0	Port pin P2 <sub>0</sub>	PDL_IO_PORT_6_3	Port pin P6 <sub>3</sub>	PDL_IO_PORT_B_3	Port pin PB <sub>3</sub>
PDL_IO_PORT_2_1	Port pin P2 <sub>1</sub>	PDL_IO_PORT_6_4	Port pin P6 <sub>4</sub>	PDL_IO_PORT_B_4	Port pin PB <sub>4</sub>
PDL_IO_PORT_2_2	Port pin P2 <sub>2</sub>	PDL_IO_PORT_6_5	Port pin P6 <sub>5</sub>	PDL_IO_PORT_B_5	Port pin PB <sub>5</sub>
PDL_IO_PORT_2_3	Port pin P2 <sub>3</sub>			PDL_IO_PORT_B_6	Port pin PB <sub>6</sub>
PDL_IO_PORT_2_4	Port pin P2 <sub>4</sub>	PDL_IO_PORT_7_0	Port pin P7 <sub>0</sub>	PDL_IO_PORT_B_7	Port pin PB <sub>7</sub>
		PDL_IO_PORT_7_1	Port pin P7 <sub>1</sub>		
PDL_IO_PORT_3_0	Port pin P3 <sub>0</sub>	PDL_IO_PORT_7_2	Port pin P7 <sub>2</sub>	PDL_IO_PORT_D_0	Port pin PD <sub>0</sub>
PDL_IO_PORT_3_1	Port pin P3 <sub>1</sub>	PDL_IO_PORT_7_3	Port pin P7 <sub>3</sub>	PDL_IO_PORT_D_1	Port pin PD <sub>1</sub>
PDL_IO_PORT_3_2	Port pin P3 <sub>2</sub>	PDL_IO_PORT_7_4	Port pin P7 <sub>4</sub>	PDL_IO_PORT_D_2	Port pin PD <sub>2</sub>
PDL_IO_PORT_3_3	Port pin P3 <sub>3</sub>	PDL_IO_PORT_7_5	Port pin P7 <sub>5</sub>	PDL_IO_PORT_D_3	Port pin PD <sub>3</sub>
		PDL_IO_PORT_7_6	Port pin P7 <sub>6</sub>	PDL_IO_PORT_D_4	Port pin PD <sub>4</sub>
PDL_IO_PORT_4_0	Port pin P4 <sub>0</sub>			PDL_IO_PORT_D_5	Port pin PD <sub>5</sub>
PDL_IO_PORT_4_1	Port pin P4 <sub>1</sub>	PDL_IO_PORT_8_0	Port pin P8 <sub>0</sub>	PDL_IO_PORT_D_6	Port pin PD <sub>6</sub>
PDL_IO_PORT_4_2	Port pin P4 <sub>2</sub>	PDL_IO_PORT_8_1	Port pin P8 <sub>1</sub>	PDL_IO_PORT_D_7	Port pin PD <sub>7</sub>
PDL_IO_PORT_4_3	Port pin P4 <sub>3</sub>	PDL_IO_PORT_8_2	Port pin P8 <sub>2</sub>		
PDL_IO_PORT_4_4	Port pin P4 <sub>4</sub>			PDL_IO_PORT_E_0	Port pin PE <sub>0</sub>
PDL_IO_PORT_4_5	Port pin P4 <sub>5</sub>	PDL_IO_PORT_9_0	Port pin P9 <sub>0</sub>	PDL_IO_PORT_E_1	Port pin PE <sub>1</sub>
PDL_IO_PORT_4_6	Port pin P4 <sub>6</sub>	PDL_IO_PORT_9_1	Port pin P9 <sub>1</sub>	PDL_IO_PORT_E_2	Port pin PE <sub>2</sub>
PDL_IO_PORT_4_7	Port pin P4 <sub>7</sub>	PDL_IO_PORT_9_2	Port pin P9 <sub>2</sub>	PDL_IO_PORT_E_3	Port pin PE <sub>3</sub>
		PDL_IO_PORT_9_3	Port pin P9 <sub>3</sub>	PDL_IO_PORT_E_4	Port pin PE <sub>4</sub>
PDL_IO_PORT_5_0	Port pin P5 <sub>0</sub>	PDL_IO_PORT_9_4	Port pin P9 <sub>4</sub>	PDL_IO_PORT_E_5	Port pin PE <sub>5</sub>
PDL_IO_PORT_5_1	Port pin P5 <sub>1</sub>	PDL_IO_PORT_9_5	Port pin P9 <sub>5</sub>		
PDL_IO_PORT_5_2	Port pin P5 <sub>2</sub>	PDL_IO_PORT_9_6	Port pin P9 <sub>6</sub>	PDL_IO_PORT_G_0	Port pin PG <sub>0</sub>
PDL_IO_PORT_5_3	Port pin P5 <sub>3</sub>			PDL_IO_PORT_G_1	Port pin PG <sub>1</sub>
PDL_IO_PORT_5_4	Port pin P5 <sub>4</sub>	PDL_IO_PORT_A_0	Port pin PA <sub>0</sub>	PDL_IO_PORT_G_2	Port pin PG <sub>2</sub>
PDL_IO_PORT_5_5	Port pin P5 <sub>5</sub>	PDL_IO_PORT_A_1	Port pin PA <sub>1</sub>	PDL_IO_PORT_G_3	Port pin PG <sub>3</sub>
		PDL_IO_PORT_A_2	Port pin PA <sub>2</sub>	PDL_IO_PORT_G_4	Port pin PG <sub>4</sub>
		PDL_IO_PORT_A_3	Port pin PA <sub>3</sub>	PDL_IO_PORT_G_5	Port pin PG <sub>5</sub>
		PDL_IO_PORT_A_4	Port pin PA <sub>4</sub>		
		PDL_IO_PORT_A_5	Port pin PA <sub>5</sub>		

Note: Refer to the hardware manual for the port pins which are available on the device that you have selected.

## 1) R\_IO\_PORT\_Set

**Synopsis**

Configure an I/O port.

**Prototype**

```
bool R_IO_PORT_Set(
    uint16_t data1, // Port pin selection
    uint8_t data2   // Configuration
);
```

**Description (1/2)**

Set the operating conditions for I/O port pins.

**[data1]**

Select the port pins to be configured (from §4.2.3). Do not use any whole-port definitions. Multiple pins on the same port may be specified, using "|" to separate each pin.

**[data2]**

Choose the pin settings.

- Direction

PDL_IO_PORT_INPUT or PDL_IO_PORT_OUTPUT
--

Input or output.

- Input buffer

PDL_IO_PORT_INPUT_BUFFER_ON or PDL_IO_PORT_INPUT_BUFFER_OFF
--

On or off.

**Return value**

True if all parameters are valid and exclusive; otherwise false.

**Category**

I/O port

**References**

R\_IO\_PORT\_ModifyControl, R\_IO\_PORT\_ReadControl

**Remarks**

- Ensure that the specified functions are valid for the selected port pin.
- The data direction and input buffer registers may be modified by other driver Create functions. Take care to not overwrite existing settings.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_io_port.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Set up port P93 as an input port */
    R_IO_PORT_Set(
        PDL_IO_PORT_9_3,
        PDL_IO_PORT_INPUT | PDL_IO_PORT_INPUT_BUFFER_ON
    );
}
```



## 2) R\_IO\_PORT\_ReadControl

### Synopsis

Read an I/O port's control register.

### Prototype

```
bool R_IO_PORT_ReadControl(
    uint16_t data1, // Port or port pin selection
    uint8_t data2,  // Control register selection
    uint8_t * data3 // Data storage location
);
```

### Description

Read an I/O port pin control setting.

#### [data1]

Use either one of the following definition values (from §4.2.3).

- One port definition or
- One port pin definition.

#### [data2]

- Select the register to be read.

PDL_IO_PORT_DIRECTION or	Data direction register.
PDL_IO_PORT_INPUT_BUFFER	Input buffer control register.

#### [data3]

The address to where the register value shall be stored.

The value will be between 0x00 and 0xFF for a port; 0 or 1 for a pin.

### Return value

True if all parameters are valid and exclusive; otherwise false.

### Category

I/O port

### References

R\_IO\_PORT\_Set, R\_IO\_PORT\_ModifyControl

### Remarks

- Ensure that the specified register is valid for the selected port pin.

### Program example

```
/* RPDL definitions */
#include "r_pdl_io_port.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    uint8_t direction;
    uint8_t output;

    /* Read the direction register for port PC */
    R_IO_PORT_ReadControl(
        PDL_IO_PORT_C,
        PDL_IO_PORT_DIRECTION
        &direction
    );

    /* Read the input buffer control for pin P03 */
    R_IO_PORT_ReadControl(
        PDL_IO_PORT_0_3,
        PDL_IO_PORT_INPUT_BUFFER
        &output
    );
}
```

### 3) R\_IO\_PORT\_ModifyControl

#### Synopsis

Modify an I/O port's control registers.

#### Prototype

```
bool R_IO_PORT_ModifyControl(
    uint16_t data1, // Port or port pin selection
    uint8_t data2,  // Control register and logical operation selection
    uint8_t data3   // Modification value
);
```

#### Description

Modifying the operation of an I/O port or I/O port pin.

##### [data1]

Use either one of the following definition values (from §4.2.3).

- One port definition or
- One port pin definition.

##### [data2]

Select the register to be modified and the logical operation, using “|” to separate the selections.

- The control register to be modified.

PDL_IO_PORT_DIRECTION or	Data direction register.
PDL_IO_PORT_INPUT_BUFFER	Input buffer control register.

- The logical operation to be applied to the control register.

PDL_IO_PORT_AND or PDL_IO_PORT_OR or PDL_IO_PORT_XOR	Select between AND (&), OR ( ) or Exclusive-OR (^).
--	---

##### [data3]

The value to be used for the modification; Between 0x00 and 0xFF for a port, 0 or 1 for a pin.

#### Return value

True if all parameters are valid and exclusive; otherwise false.

#### Category

I/O port

#### References

R\_IO\_PORT\_Set, R\_IO\_PORT\_ReadControl

#### Remarks

- Ensure that the specified functions are valid for the selected port pin.
- The data direction and input buffer registers may be modified by other driver Create functions. Take care to not overwrite existing settings.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_io_port.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Set the lower 4 bits on port P1 to output */
    R_IO_PORT_ModifyControl(
        PDL_IO_PORT_1,
        PDL_IO_PORT_DIRECTION | PDL_IO_PORT_OR,
        0x0F
    );

    /* Enable the pull-up on pin PA3 */
    R_IO_PORT_ModifyControl(
        PDL_IO_PORT_A_3,
        PDL_IO_PORT_INPUT_BUFFER | PDL_IO_PORT_OR,
        1
    );
}
```

#### 4) R\_IO\_PORT\_Read

##### Synopsis

Read data from an I/O port.

##### Prototype

```
bool R_IO_PORT_Read(
    uint16_t data1, // Port or port pin selection
    uint8_t * data2 // Pointer to the variable in which the value shall be stored.
);
```

##### Description

Gets the value of an I/O port or I/O port pin.

##### [data1]

Use either one of the following definition values (from §4.2.3).

- One port definition or
- One port pin definition.

##### [data2]

The value will be between 0x00 and 0xFF for a port, 0 or 1 for a pin.

##### Return value

If the I/O port specification is incorrect, false is returned; otherwise, true is returned.

##### Category

I/O port

##### Reference

R\_IO\_PORT\_Set

##### Remark

- If an invalid port or pin is specified, the operation of the function cannot be guaranteed.
- For correct reading of pins on ports 4, 5 and 6, the input buffer must be switched on.

##### Program example

```
/* RPD_L definitions */
#include "r_pdl_io_port.h"

/* RPD_L device-specific definitions */
#include "r_pdl_definitions.h"

void func( void )
{
    uint8_t data;

    /* Get the value of port pin P12 */
    R_IO_PORT_Read(
        PDL_IO_PORT_1_2,
        &data
    );

    /* Get the value of port 4 */
    R_IO_PORT_Read(
        PDL_IO_PORT_4,
        &data
    );
}
```

## 5) R\_IO\_PORT\_Write

### Synopsis

Write data to an I/O port.

### Prototype

```
bool R_IO_PORT_Write(
    uint16_t data1, // Port or port pin selection
    uint8_t data2   // The data to be written to the I/O port or port pin.
);
```

### Description

Write data to an I/O port or I/O port pin.

#### [data1]

Use either one of the following definition values (from §4.2.3).

- One port definition or
- One port pin definition.

#### [data2]

The value must be between 0x00 and 0xFF for a port, 0 or 1 for a pin.

### Return value

True if the parameters are valid; otherwise false.

### Category

I/O port

### References

R\_IO\_PORT\_Set, R\_IO\_PORT\_Read

### Remarks

- If an invalid port or pin is specified, the operation of the function cannot be guaranteed.

### Program example

```
/* RPDL definitions */
#include "r_pdl_io_port.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func( void )
{
    /* Set the output of port pin P05 */
    R_IO_PORT_Write(
        PDL_IO_PORT_0_5,
        0
    );

    /* Set the output of port 6 */
    R_IO_PORT_Write(
        PDL_IO_PORT_6,
        0x55
    );
}
```

## 6) R\_IO\_PORT\_Compare

### Synopsis

Check the pin states on an I/O port.

### Prototype

```
bool R_IO_PORT_Compare(
    uint16_t data1, // Input port or port pin selection
    uint8_t data2,  // Comparison value
    void * func     // Function pointer
);
```

### Description

Read the input state of an I/O port or I/O port pin and call a function if a match occurs.

#### [data1]

Use either one of the following definition values (from §4.2.3):

- One port definition or
- One port pin definition.

#### [data2]

The value to be compared with; Between 0x00 and 0xFF for a port, 0 or 1 for a pin.

#### [func]

The function to be called if a match occurs.

### Return value

True if the parameters are valid; otherwise false.

### Category

I/O port

### References

R\_IO\_PORT\_Set

### Remarks

- If an invalid port or pin is specified, the operation of the function cannot be guaranteed.

### Program example

```
/* RPD_L definitions */
#include "r_pdl_io_port.h"

/* RPD_L device-specific definitions */
#include "r_pdl_definitions.h"

void IoHandler1{}
void IoHandler2{}

void func( void )
{
    /* Call function IoHandler1 if port pin P11 is high */
    R_IO_PORT_Compare(
        PDL_IO_PORT_1_1,
        1,
        IoHandler1
    );

    /* Call function IoHandler2 if port 6 reads as 0x55 */
    R_IO_PORT_Compare(
        PDL_IO_PORT_6,
        0x55,
        IoHandler2
    );
}
```

## 7) R\_IO\_PORT\_Modify

### Synopsis

Modify the pin states on an I/O port.

### Prototype

```
bool R_IO_PORT_Modify(
    uint16_t data1, // Output port or port pin selection
    uint8_t data2,  // Logical operation
    uint8_t data3   // Modification value
);
```

### Description

Read the output state of an I/O port or I/O port pin, modify the result and write it back to the port.

#### [data1]

Use either one of the following definition values (from §4.2.3).

- One port definition or
- One port pin definition.

#### [data2]

- The logical operation to be applied to the port or port pin.

PDL_IO_PORT_AND or PDL_IO_PORT_OR or PDL_IO_PORT_XOR	Select between AND (&), OR ( ) or Exclusive-OR (^).
--	---

#### [data3]

The value to be used for the modification; Between 0x00 and 0xFF for a port, 0 or 1 for a pin.

### Return value

True if the parameters are valid; otherwise false.

### Category

I/O port

### References

R\_IO\_PORT\_Set, R\_IO\_PORT\_Read, R\_IO\_PORT\_Write

### Remarks

- If an invalid port or pin is specified, the operation of the function cannot be guaranteed.

### Program example

```
/* RPDL definitions */
#include "r_pdl_io_port.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func( void )
{
    /* Invert port pin P11 */
    R_IO_PORT_Modify(
        PDL_IO_PORT_1_1,
        PDL_IO_PORT_XOR,
        1
    );

    /* And the value port 6 with 0x55 */
    R_IO_PORT_Modify(
        PDL_IO_PORT_6,
        PDL_IO_PORT_AND,
        0x55
    );
}
```

## 8) R\_IO\_PORT\_Wait

### Synopsis

Wait for a match on an I/O port.

### Prototype

```
bool R_IO_PORT_Wait(
    uint16_t data1, // Output port or port pin selection
    uint8_t data2   // Comparison value
);
```

### Description

Loop until an I/O port or I/O port pin matches the comparison value.

#### [data1]

Use either one of the following definition values (from §4.2.3).

- One port definition or
- One port pin definition.

#### [data2]

The value to be compared with; Between 0x00 and 0xFF for a port, 0 or 1 for a pin.

### Return value

True if the parameters are valid; otherwise false.

### Category

I/O port

### References

R\_IO\_PORT\_Set, R\_IO\_PORT\_Read

### Remarks

- If an invalid port or pin is specified, the operation of the function cannot be guaranteed.
- This function waits for the I/O port or port pin value to match the comparison data. If the I/O port's control registers are directly modified by the user, this function may lock up.

### Program example

```
/* RPDL definitions */
#include "r_pdl_io_port.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func( void )
{
    /* Wait until pin P11 reads as 0 */
    R_IO_PORT_Wait(
        PDL_IO_PORT_1_1,
        0
    );

    /* Wait until port 6 reads as 0x55 */
    R_IO_PORT_Wait(
        PDL_IO_PORT_6,
        0x55
    );
}
```



#### 4.2.4. Port Function Control

Each I/O Port function can operate on a complete port, or on individual port pins. The definitions available to functions are listed below.

PFC register definitions

PDL_PFC_PF8IRQ
PDL_PFC_PF9IRQ
PDL_PFC_PFAADC
PDL_PFC_PFCMTU
PDL_PFC_PFDGPT
PDL_PFC_PFFSCI
PDL_PFC_PFGSPI
PDL_PFC_PFHSPI
PDL_PFC_PFJCAN
PDL_PFC_PFKLIN
PDL_PFC_PFMPOE
PDL_PFC_PFNPOE

**1) R\_PFC\_Read****Synopsis**

Read a PFC register.

**Prototype**

```
bool R_PFC_Read(
    uint8_t data1,    // PFC register selection
    uint8_t * data2   // Pointer to the variable where the PFC register's value shall be stored.
);
```

**Description**

Get the value of a PFC register.

**[data1]**

One of the definition values from §4.2.4.

**[data2]**

The value read from the register.

**Return value**

True if a valid register is specified; otherwise false.

**Category**

PFC registers

**References**

R\_PFC\_Write

**Remarks**

- None.

**Program example**

```
/* RPD_L definitions */
#include "r_pdl_pfc.h"

/* RPD_L device-specific definitions */
#include "r_pdl_definitions.h"

void func( void )
{
    uint8_t data;

    /* Get the value of register PFCMTU */
    R_PFC_Read(
        PDL_PFC_PFCMTU,
        &data
    );
}
```

**2) R\_PFC\_Write****Synopsis**

Write to a PFC register.

**Prototype**

```
bool R_PFC_Write(
    uint8_t data1, // PFC register selection
    uint8_t data2  // Data to be written to the PFC register
);
```

**Description**

Write the value to a PFC register.

**[data1]**

One of the definition values from §4.2.4.

**[data2]**

The value to be written to the register.

**Return value**

True if a valid register is specified; otherwise false.

**Category**

PFC registers

**References**

R\_PFC\_Read, R\_PFC\_Modify

**Remarks**

- The PFC registers are modified by other driver functions. Take care to not overwrite existing settings.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_pfc.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func( void )
{
    /* Write data to register PFDGPT */
    R_PFC_Write(
        PDL_PFC_PFDGPT,
        0xFF
    );
}
```

### 3) R\_PFC\_Modify

**Synopsis**

Modify a PFC register.

**Prototype**

```
bool R_PFC_Modify(
    uint8_t data1, // PFC register selection
    uint8_t data2, // Logical operation
    uint8_t data3  // Modification value
);
```

**Description**

Write the value to a PFC register.

**[data1]**

One of the definition values from §4.2.4.

**[data2]**

- The logical operation to be applied to the register contents.

PDL_PFC_AND or PDL_PFC_OR or PDL_PFC_XOR	Select between AND (&), OR ( ) or Exclusive-OR (^).
--	---

**[data3]**

The value to be used for the modification.

**Return value**

True if a valid register is specified; otherwise false.

**Category**

PFC registers

**References**

R\_PFC\_Read

**Remarks**

- The PFC registers are modified by other driver functions. Take care to not overwrite existing settings.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_pfc.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func( void )
{
    /* Set bit 7 in PFDMTU to 1 */
    R_PFC_Modify(
        PDL_PFC_PFDMTU,
        PDL_PFC_OR,
        0x80
    );
}
```

## 4.2.5. MCU operation

## 1) R\_MCU\_Control

**Synopsis**

Control the operation of the MCU.

**Prototype**

```
bool R_MCU_Control(
    uint8_t data // Control options
);
```

**Description**

Modify the MCU control registers.

**[data]**

Select the operation states. If multiple selections are required, use "|" to separate each selection. Specify PDL\_NO\_DATA to use the defaults.

- On-chip ROM control

PDL_MCU_ROM_ENABLE or PDL_MCU_ROM_DISABLE	Enable or disable the on-chip ROM.
--	------------------------------------

- On-chip RAM control

PDL_MCU_RAM_ENABLE or PDL_MCU_RAM_DISABLE	Enable or disable the on-chip RAM.
--	------------------------------------

**Return value**

True if a valid register is specified; otherwise false.

**Category**

MCU registers

**References**

R\_MCU\_GetStatus

**Remarks**

- None.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_mcu.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func( void )
{
    /* Modify the MCU operation */
    R_MCU_Control(
        PDL_MCU_ROM_DISABLE
    );
}
```

## 2) R\_MCU\_GetStatus

### Synopsis

Read the MCU status.

### Prototype

```
bool R_MCU_GetStatus(
    uint16_t* data // Pointer to the variable where the status value shall be stored.
);
```

### Description

Read the status registers for the MCU.

#### [data]

The status flags shall be stored in the format below.

b15 – b13		b12	b11 – b9	b8
0		Boot mode	0	On-chip ROM
		0: Other mode 1: Boot mode		0: Disabled 1: Enabled

b7	b6 – b2	b1	b0
Endian	0	Pin states	
0: Little 1: Big		MD1	MD0

### Return value

True.

### Category

MCU registers

### References

R\_MCU\_Control

### Remarks

- None.

### Program example

```
/* RPD_L definitions */
#include "r_pdl_mcu.h"

/* RPD_L device-specific definitions */
#include "r_pdl_definitions.h"

void func( void )
{
    uint16_t status;

    /* Read the MCU status registers */
    R_MCU_GetStatus(
        &status
    );
}
```

## 4.2.6. Low Power Consumption

## 1) R\_LPC\_Create

**Synopsis**

Configure the MCU low power conditions.

**Prototype**

```
bool R_LPC_Create(
    uint16_t data1, // Configuration options
    uint32_t data2  // Waiting times
);
```

**Description**

Load the registers that control module or CPU operation.

**[data1]**

Select the required settings. If multiple selections are required, use “|” to separate each selection.

The default settings are shown in **bold**. Specify PDL\_NO\_DATA to use the defaults.

- I/O port retention control

<b>PDL_LPC_IO_SAME</b> or PDL_LPC_IO_DELAY	Select whether IO port retention is cancelled when deep software standby mode is ended, or when CPU operation has resumed.
---	--

- Deep software standby cancel control

<b>PDL_LPC_CANCEL_IRQ0_DISABLE</b> or PDL_LPC_CANCEL_IRQ0_FALLING or PDL_LPC_CANCEL_IRQ0_RISING	Prevent or allow an edge on the IRQ0-A pin to cancel deep software standby mode.
<b>PDL_LPC_CANCEL_IRQ1_DISABLE</b> or PDL_LPC_CANCEL_IRQ1_FALLING or PDL_LPC_CANCEL_IRQ1_RISING	Prevent or allow an edge on the IRQ1-A pin to cancel deep software standby mode.
<b>PDL_LPC_CANCEL_LVD_DISABLE</b> or PDL_LPC_CANCEL_LVD_ENABLE	Prevent or allow the voltage detection circuit to cancel deep software standby mode.
<b>PDL_LPC_CANCEL_NMI_DISABLE</b> or PDL_LPC_CANCEL_NMI_FALLING or PDL_LPC_CANCEL_NMI_RISING	Prevent or allow an edge on the NMI pin to cancel deep software standby mode.

**[data2]**Select the waiting times. If multiple selections are required, use “|” to separate each selection. The default settings are shown in **bold**. Specify PDL\_NO\_DATA to use the defaults.

- Software Standby waiting time

PDL_LPC_STANDBY_64 or PDL_LPC_STANDBY_512 or PDL_LPC_STANDBY_1024 or PDL_LPC_STANDBY_2048 or PDL_LPC_STANDBY_4096 or PDL_LPC_STANDBY_16384 or PDL_LPC_STANDBY_32768 or PDL_LPC_STANDBY_65536 or PDL_LPC_STANDBY_131072 or PDL_LPC_STANDBY_262144 or <b>PDL_LPC_STANDBY_524288</b>	Select the number of PCLK cycles that will elapse before the CPU resumes after exiting from software standby mode.
---	--

- Deep Software Standby waiting time

PDL_LPC_DEEP_STANDBY_64 or PDL_LPC_DEEP_STANDBY_512 or PDL_LPC_DEEP_STANDBY_1024 or PDL_LPC_DEEP_STANDBY_2048 or PDL_LPC_DEEP_STANDBY_4096 or PDL_LPC_DEEP_STANDBY_16384 or PDL_LPC_DEEP_STANDBY_32768 or PDL_LPC_DEEP_STANDBY_65536 or PDL_LPC_DEEP_STANDBY_131072 or PDL_LPC_DEEP_STANDBY_262144 or <b>PDL_LPC_DEEP_STANDBY_524288</b>	Select the number of PCLK cycles that will elapse before the CPU resumes after exiting from deep software standby mode.
--	---

<b>Return value</b>	True if all parameters are valid and exclusive; otherwise false.
<b>Category</b>	Low Power Consumption
<b>References</b>	R_LPC_Control
<b>Remarks</b>	<ul style="list-style-type: none"><li>• If PDL_LPC_IO_DELAY is specified, use R_LPC_Control with the PDL_LPC_IO_RELEASE option to cancel the I/O port state retention.</li></ul>
<b>Program example</b>	<pre>/* RPD_L definitions */ #include "r_pdl_lpc.h"  /* RPD_L device-specific definitions */ #include "r_pdl_definitions.h"  void func( void ) {     /* Allow a falling edge on IRQ0-A to cancel deep software standby */     R_LPC_Create(         PDL_LPC_CANCEL_IRQ0_FALLING,         PDL_LPC_STANDBY_64   PDL_LPC_DEEP_STANDBY_1024     ); }</pre>



## 2) R\_LPC\_Control

### Synopsis

Select a low power consumption mode.

### Prototype

```
bool R_LPC_Control(
    uint8_t data    // Mode selection
);
```

### Description

Transition to one of the low power modes.

#### [data]

Control selection. All selections are optional.

If multiple selections are required, use "|" to separate each selection.

#### • Mode selection

PDL_LPC_MODE_SLEEP or PDL_LPC_MODE_ALL_MODULE_CLOCK_STOP or PDL_LPC_MODE_SOFTWARE_STANDBY or PDL_LPC_MODE_DEEP_SOFTWARE_STANDBY
--

Select the mode to be entered.

#### • I/O port retention cancellation

PDL_LPC_IO_RELEASE
--------------------

Cancel the retention of I/O port pin states.
--

### Return value

True if all parameters are valid and exclusive; otherwise false.

### Category

Low Power Consumption

### References

R\_LPC\_Create

### Remarks

- Sleep mode is utilised by some peripheral drivers to turn off the CPU when required.
- When entering software standby or deep software standby mode, the oscillation stop detection function is disabled. The detection is re-enabled if software standby mode is interrupted.
- On exit from deep software standby mode, the MCU is reset.
- The peripheral Create functions bring modules out of the clock-stop state as required. The peripheral Destroy functions put modules into the clock-stop state as required. When All Module Clock-Stop mode is cancelled, the peripherals that were active when that mode was entered will be re-activated.

### Program example

```
/* RPDL definitions */
#include "r_pdl_lpc.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func( void )
{
    /* Enter deep software standby mode */
    R_LPC_Control(
        PDL_LPC_MODE_DEEP_SOFTWARE_STANDBY
    );

    /* Clear the I/O port state retention */
    R_LPC_Control(
        PDL_LPC_IO_RELEASE
    );
}
```

### 3) R\_LPC\_WriteBackup

#### Synopsis

Write to the Backup registers.

#### Prototype

```
bool R_LPC_WriteBackup(
    uint8_t * data1,    // Data pointer
    uint8_t data2       // Data count
);
```

#### Description

Load the registers that control module or CPU operation.

#### [data1]

The data to be written to the backup area.

#### [data2]

The number of bytes to be written to the backup area. Valid from 1 to 32.

#### Return value

True if all parameters are valid; otherwise false.

#### Category

Low Power Consumption

#### References

R\_LPC\_ReadBackup

#### Remarks

- The definition R\_PDL\_LPC\_BACKUP\_AREA\_SIZE specifies the number of bytes that are available

#### Program example

```
/* RPD_L definitions */
#include "r_pdl_lpc.h"

/* RPD_L device-specific definitions */
#include "r_pdl_definitions.h"

void func( void )
{
    uint8_t data_to_save[R_PDL_LPC_BACKUP_AREA_SIZE];

    /* Write data into the backup registers */
    R_LPC_WriteBackup(
        data_to_save,
        R_PDL_LPC_BACKUP_AREA_SIZE
    );
}
```

#### 4) R\_LPC\_ReadBackup

##### Synopsis

Read from the Backup registers.

##### Prototype

```
bool R_LPC_ReadBackup(
    uint8_t * data1, // Data pointer
    uint8_t data2    // Data count
);
```

##### Description

Load the registers that control module or CPU operation.

##### [data1]

The storage area for the data read from the backup area.

##### [data2]

The number of bytes to be read from the backup area. Valid from 1 to 32.

##### Return value

True if all parameters are valid; otherwise false.

##### Category

Low Power Consumption

##### References

R\_LPC\_WriteBackup

##### Remarks

- The definition R\_PDL\_LPC\_BACKUP\_AREA\_SIZE specifies the number of bytes that are available

##### Program example

```
/* RPD_L definitions */
#include "r_pdl_lpc.h"

/* RPD_L device-specific definitions */
#include "r_pdl_definitions.h"

void func( void )
{
    uint8_t data_to_restore[R_PDL_LPC_BACKUP_AREA_SIZE];

    /* Read data from the backup registers */
    R_LPC_ReadBackup(
        data_to_restore,
        R_PDL_LPC_BACKUP_AREA_SIZE
    );
}
```

## 5) R\_LPC\_GetStatus

### Synopsis

Read the status flags.

### Prototype

```
bool R_LPC_GetStatus(
    uint16_t * data    // Data pointer
);
```

### [data]

The status flags shall be stored in the format below.

b15				b14 – b11		b10	b9	b8
Event detection flags (0: not detected; 1: detected)								
An interrupt has caused an exit from deep software standby mode, followed by an internal reset				0		LVD2	LVD1	Power-on reset

b7	b6	b5	b4	b3	b2	b1	b0
Deep Software Standby cancel request detection							
0: No activity							
1: The exit from deep software standby was caused by trigger signal on the following pins.							
NMI	0	0	LVD	0	0	IRQ1-A	IRQ0-A

### Return value

True.

### Category

Low Power Consumption

### References

R\_LPC\_Create, R\_LPC\_Control

### Remarks

- If a flag is set to 1, it shall be automatically cleared to 0 by this function (apart from the Power-on reset flag, which can be cleared only by a hardware reset).

### Program example

```
/* RPD_L definitions */
#include "r_pdl_lpc.h"

/* RPD_L device-specific definitions */
#include "r_pdl_definitions.h"

void func( void )
{
    uint16_t status_flags;

    /* Find out what caused the exit from deep software standby */
    R_LPC_GetStatus(
        &status_flags
    );
}
```

## 4.2.7. Voltage Detection Circuit

## 1) R\_LVD\_Control

**Synopsis**

Configure the voltage detection circuit.

**Prototype**

```
bool R_LVD_Control(
    uint8_t data // Configuration selection
);
```

**Description**

Set the voltage detection configuration.

**[data]**

Set the voltage detection operation.

- Detection configuration

PDL_LVD_VDET2_DISABLE_VDET1_DISABLE or PDL_LVD_VDET2_DISABLE_VDET1_RESET or PDL_LVD_VDET2_DISABLE_VDET1_INTERRUPT or PDL_LVD_VDET2_RESET_VDET1_DISABLE or PDL_LVD_VDET2_INTERRUPT_VDET1_DISABLE or PDL_LVD_VDET2_INTERRUPT_VDET1_RESET	Select whether the detection of the supply voltage $V_{CC}$ below levels $V_{det2}$ and $V_{det1}$ is ignored, causes a reset or generates an interrupt request.
---	--

**Return value**

True if the parameter is valid; otherwise false.

**Category**

Voltage detection circuit

**References**

R\_INTC\_CreateExtInterrupt, R\_INTC\_GetExtInterruptStatus

**Remarks**

- Use R\_INTC\_CreateExtInterrupt and R\_INTC\_GetExtInterruptStatus to monitor LVD interrupt requests.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_lvd.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func( void )
{
    /* Generate an NMI when Vdet2 is reached; reset if Vdet1 is reached */
    /*
    R_LVD_Control(
        PDL_LVD_VDET2_INTERRUPT_VDET1_RESET
    );
    */
}
```

## 4.2.8. Bus Controller

## 1) R\_BSC\_Create

**Synopsis**

Configure the bus controller.

**Prototype**

```
bool R_BSC_Create(
    uint8_t data1,    // Configuration
    void * func,      // Callback function
    uint8_t data2     // Interrupt priority level
);
```

**Description (1/2)**

Configure the error detection and register the callback function

Control the bus controller.

The default setting is shown in **bold**. Specify PDL\_NO\_DATA to use the default.**[data1]**

- Error monitoring

PDL_BSC_ERROR_ILLEGAL_ADDRESS_ENABLE or <b>PDL_BSC_ERROR_ILLEGAL_ADDRESS_DISABLE</b>	Enable or disable illegal address access detection.
---	--

**[func]**

The function to be called when a bus error occurs. Specify PDL\_NO\_FUNC if not required.

**[data2]**

The interrupt priority level. Select between 1 (lowest priority) and 15 (highest priority).

This parameter will be ignored if PDL\_NO\_FUNC is specified for parameter func.

**Return value**

True if all parameters are valid and exclusive; otherwise false.

**Category**

Bus Controller

**Reference**

R\_BSC\_Control, R\_BSC\_GetStatus

**Remarks**

- A callback function is executed by the interrupt processing function. This means that no other interrupt can be processed until the callback function has completed.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_bsc.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

/* Bus error handler */
void BusErrorFunc(void){}

void func(void)
{
    /* Enable interrupts and register the callback function */
    R_BSC_Create(
        PDL_BSC_ERROR_ILLEGAL_ADDRESS_ENABLE,
        BusErrorFunc,
        5
    );
}
```

## 2) R\_BSC\_Control

### Synopsis

Modify the Bus Controller operation.

### Prototype

```
bool R_BSC_Control(
    uint8_t data    // Control options
);
```

### Description

Provide interrupt flag clearing

#### [data]

- Error clearing

PDL_BSC_ERROR_CLEAR	Clear the bus-error status registers.
PDL_BSC_DISABLE_BUSERR_IRQ	Disable bus error interrupt request.

### Return value

True if all parameters are valid; otherwise false.

### Category

Bus Controller

### Reference

R\_BSC\_Create, R\_BSC\_GetStatus

### Remarks

- This function can be called from the error handling function (see R\_BSC\_Create).

### Program example

```
/* RPDL definitions */
#include "r_pdl_bsc.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Clear the bus error signals */
    R_BSC_Control(
        PDL_BSC_ERROR_CLEAR
    );
}
```

### 3) R\_BSC\_GetStatus

#### Synopsis

Read the Bus Controller status flags.

#### Prototype

```
bool R_BSC_GetStatus(
    uint8_t * data1,    // A pointer to the data storage location
    uint16_t * data2    // A pointer to the data storage location
);
```

#### Description

Read the interrupt status

#### [data1]

The status flags shall be stored in the following format.  
Specify PDL\_NO\_PTR if the status flags are not required.

b7	b6 – b4	b3 – b1	b0
0	000b: CPU access 011b: DTC access	0	0: Valid address access 1: Illegal address detected.

#### [data2]

The address that was accessed illegally. Specify PDL\_NO\_PTR if the address is not required.

b15 – b3	b2 – b0
The upper 13 bits of the address that was accessed illegally.	0

#### Return value

True.

#### Category

Bus Controller

#### Reference

R\_BSC\_Create, R\_BSC\_Control

#### Remarks

- Use R\_BSC\_Control to clear the flags and address.

#### Program example

```
/* RPDL definitions */
#include "r_pdl_bsc.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    uint8_t status;
    uint16_t bad_address;

    /* Read the flags */
    R_BSC_GetStatus(
        &status,
        &bad_address
    );
}
```



## 4.2.9. Data Transfer Controller

## 1) R\_DTC\_Set

**Synopsis**

Set the Data Transfer Controller options.

**Prototype**

```
bool R_DTC_Set (
    uint8_t data1,    // Configuration options
    uint32_t * data2  // Vector table base address
);
```

**Description**

Set the global options for the Data Transfer Controller.

**[data1]**

Configuration selections.

If multiple selections are required, use "|" to separate each selection.

The default settings are shown in **bold**. Specify PDL\_NO\_DATA to use the defaults.

- Read skip control

<b>PDL_DTC_READ_SKIP_DISABLE</b> or PDL_DTC_READ_SKIP_ENABLE	Disable or enable skipping of transfer data read when the vector numbers match.
---	---

- Address size control

<b>PDL_DTC_ADDRESS_FULL</b> or PDL_DTC_ADDRESS_SHORT	Select 32-bit (full) or 24-bit (short) address mode.
---	--

**[data2]**

The first address of the area of on-chip RAM where the DTC vector table shall be stored.

The address must be on a 4 kB boundary i.e. have the format xxxxx000h.

**Return value**

True if all parameters are valid and exclusive; otherwise false.

**Category**

Data Transfer Controller

**Reference**

R\_DTC\_Create

**Remarks**

- Before calling R\_DTC\_Create, call this function once.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_dtc.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

/* Reserve an area for the DTC vector table */
#pragma address dtc_vector_table = 0x00001000
uint32_t dtc_vector_table[256];

void func(void)
{
    /* Configure the controller */
    R_DTC_Set(
        PDL_DTC_ADDRESS_SHORT,
        dtc_vector_table
    );
}
```

## 2) R\_DTC\_Create

### Synopsis

Configure the Data Transfer Controller for a transfer.

### Prototype

```
bool R_DTC_Create(
    uint32_t data1,    // Configuration selection
    uint32_t * data2,  // Transfer data start address
    void * data3,      // Source start address
    void * data4,      // Destination start address
    uint16_t data5,    // Transfer count
    uint8_t data6      // Block size
);
```

### Description (1/3)

Configure DTC activation for one trigger source.

#### [data1]

Configuration selections.

If multiple selections are required, use “|” to separate each selection.

The default settings are shown in **bold**.

- Transfer mode selection

PDL_DTC_NORMAL or PDL_DTC_REPEAT or PDL_DTC_BLOCK	Normal or Repeat or Block mode.
PDL_DTC_SOURCE or PDL_DTC_DESTINATION	If Repeat or Block mode is selected, select the source or destination side to be the Repeat or Block area.

- Address direction selection

PDL_DTC_SOURCE_ADDRESS_FIXED or PDL_DTC_SOURCE_ADDRESS_PLUS or PDL_DTC_SOURCE_ADDRESS_MINUS	After a data transfer, leave the source address unchanged, increment it or decrement it.
PDL_DTC_DESTINATION_ADDRESS_FIXED or PDL_DTC_DESTINATION_ADDRESS_PLUS or PDL_DTC_DESTINATION_ADDRESS_MINUS	After a data transfer, leave the destination address unchanged, increment it or decrement it.

- Transfer data size

PDL_DTC_SIZE_8 or PDL_DTC_SIZE_16 or PDL_DTC_SIZE_32	Select 1, 2 or 4 bytes to be transferred in one operation.
--	---

- Chain transfer control

<b>PDL_DTC_CHAIN_DISABLE</b> or PDL_DTC_CHAIN_CONTINUOUS or PDL_DTC_CHAIN_0	Disable, Enable continuous or Enable only when the transfer counter is 0.
---	---

- Interrupt generation

<b>PDL_DTC_IRQ_COMPLETE</b> or PDL_DTC_IRQ_TRANSFER	Select interrupt request generation when the transfer sequence completes, or for every transfer.
--	---

- Trigger selection

Name	Trigger cause
PDL_DTC_TRIGGER_CHAIN or	Chain transfer.
PDL_DTC_TRIGGER_SW or	By software.
PDL_DTC_TRIGGER_CMT0 or PDL_DTC_TRIGGER_CMT1 or PDL_DTC_TRIGGER_CMT2 or PDL_DTC_TRIGGER_CMT3 or	Compare match on channel CMTn (n = 0 to 3).
PDL_DTC_TRIGGER_SPI0_RX or	Receive buffer full on SPI channel 0.
PDL_DTC_TRIGGER_SPI0_TX or	Transmit buffer empty on RSPI channel 0.

Description (2/3)	
PDL_DTC_TRIGGER_IRQ0 or PDL_DTC_TRIGGER_IRQ1 or PDL_DTC_TRIGGER_IRQ2 or PDL_DTC_TRIGGER_IRQ3 or PDL_DTC_TRIGGER_IRQ4 or PDL_DTC_TRIGGER_IRQ5 or PDL_DTC_TRIGGER_IRQ6 or PDL_DTC_TRIGGER_IRQ7 or	Valid edge detected on pin IRQn (n = 0 to 7).
PDL_DTC_TRIGGER_AD10 or	Conversion completed on the 10-bit ADC unit.
PDL_DTC_TRIGGER_S12AD10 or PDL_DTC_TRIGGER_S12AD11 or	Conversion completed on the 12-bit ADC unit n (n = 0 to 1).
PDL_DTC_TRIGGER_CMPI or PDL_DTC_TRIGGER_TGIA0 or PDL_DTC_TRIGGER_TGIA1 or PDL_DTC_TRIGGER_TGIA2 or PDL_DTC_TRIGGER_TGIA3 or PDL_DTC_TRIGGER_TGIA4 or PDL_DTC_TRIGGER_TGIA6 or PDL_DTC_TRIGGER_TGIA7 or	Comparison condition met.
PDL_DTC_TRIGGER_TGIB0 or PDL_DTC_TRIGGER_TGIB1 or PDL_DTC_TRIGGER_TGIB2 or PDL_DTC_TRIGGER_TGIB3 or PDL_DTC_TRIGGER_TGIB4 or PDL_DTC_TRIGGER_TGIB6 or PDL_DTC_TRIGGER_TGIB7 or	Compare match or input capture A on MTU channel n (n = 0 to 4 or 6 to 7).
PDL_DTC_TRIGGER_TGIC0 or PDL_DTC_TRIGGER_TGIC3 or PDL_DTC_TRIGGER_TGIC4 or PDL_DTC_TRIGGER_TGIC6 or PDL_DTC_TRIGGER_TGIC7 or PDL_DTC_TRIGGER_TGID0 or PDL_DTC_TRIGGER_TGID3 or PDL_DTC_TRIGGER_TGID4 or PDL_DTC_TRIGGER_TGID6 or PDL_DTC_TRIGGER_TGID7 or	Compare match or input capture B on MTU channel n (n = 0 to 4 or 6 to 7).
PDL_DTC_TRIGGER_TGIU5 or PDL_DTC_TRIGGER_TGIV5 or PDL_DTC_TRIGGER_TGIW5 or	Compare match or input capture C on MTU channel n (n = 0, 3, 4, 6 or 7).
PDL_DTC_TRIGGER_TCIV4 or PDL_DTC_TRIGGER_TCIV7 or	Compare match or input capture D on MTU channel n (n = 0, 3, 4, 6 or 7).
PDL_DTC_TRIGGER_GTCIA0 or PDL_DTC_TRIGGER_GTCIA1 or PDL_DTC_TRIGGER_GTCIA2 or PDL_DTC_TRIGGER_GTCIA3 or PDL_DTC_TRIGGER_GTCIB0 or PDL_DTC_TRIGGER_GTCIB1 or PDL_DTC_TRIGGER_GTCIB2 or PDL_DTC_TRIGGER_GTCIB3 or PDL_DTC_TRIGGER_GTCIC0 or PDL_DTC_TRIGGER_GTCIC1 or PDL_DTC_TRIGGER_GTCIC2 or PDL_DTC_TRIGGER_GTCIC3 or PDL_DTC_TRIGGER_GTCIE0 or PDL_DTC_TRIGGER_GTCIE1 or PDL_DTC_TRIGGER_GTCIE2 or PDL_DTC_TRIGGER_GTCIE3 or PDL_DTC_TRIGGER_GTCIV0 or PDL_DTC_TRIGGER_GTCIV1 or PDL_DTC_TRIGGER_GTCIV2 or PDL_DTC_TRIGGER_GTCIV3 or	Compare match or input capture U on MTU channel 5. Compare match or input capture V on MTU channel 5. Compare match or input capture W on MTU channel 5
PDL_DTC_TRIGGER_LOCO1 or	Counter overflow or underflow on MTU channel n (n = 4 or 7).
	Compare match or input capture A on GPT channel n (n = 0 to 3).
	Compare match or input capture B on GPT channel n (n = 0 to 3).
	Compare match or input capture C or D on GPT channel n (n = 0 to 3).
	Compare match or input capture E or F on GPT channel n (n = 0 to 3).
	Counter limit match on GPT channel n (n = 0 to 3).
	LOCO count function event.

<b>Description (3/3)</b>	PDL_DTC_TRIGGER_RXI0 or PDL_DTC_TRIGGER_RXI1 or PDL_DTC_TRIGGER_RXI2 or PDL_DTC_TRIGGER_TXI0 or PDL_DTC_TRIGGER_TXI1 or PDL_DTC_TRIGGER_TXI2 or	Receive buffer full on SCI channel n (n = 0 to 2).
	PDL_DTC_TRIGGER_ICRXI0 or PDL_DTC_TRIGGER_ICTXI0	Transmit buffer empty on SCI channel n (n = 0 to 2).
		Receive buffer full on I <sup>2</sup> C channel 0.
		Transmit buffer empty on I <sup>2</sup> C channel 0.

**[data2]**

The start address of the transfer data area. It must be a multiple of 4.  
For short address mode, 12 bytes are required to store the transfer data.  
For full address mode, 16 bytes are required.

**[data3]**

The source start address. The valid range depends on the address mode (short or full).

**[data4]**

The destination start address. The valid range depends on the address mode (short or full).

**[data5]**

The number of transfers to take place.  
For normal or block mode, valid between 0 and 65535 (0 = 65536 transfers).  
For repeat mode, valid between 0 and 255 (0 = 256 transfers).

**[data6]**

The block size for each transfer. Valid between 1 and 255 units.  
Ignored in normal or repeat mode.

<b>Return value</b>	True if all parameters are valid and exclusive; otherwise false.
<b>Category</b>	Data Transfer Controller
<b>Reference</b>	R_DTC_Set, R_DTC_Control
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• If address increment or decrement is selected, the address changes according to the number of bytes (1, 2 or 4) in each transfer.</li> <li>• Before calling this function, call R_DTC_Set.</li> <li>• Call this function before configuring the peripherals that will be involved in the data transfer.</li> <li>• Call this function once for each peripheral that will trigger a transfer, and for each chained transfer.</li> <li>• When all calls to this function are complete, call R_DTC_Control to start the DTC.</li> </ul>

**Program example**

```
/* RPDL definitions */
#include "r_pdl_dtc.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

/* Reserve 16 bytes (full address mode) for the CMT0-triggered transfer
data area */
/* Use a 32-bit type to make the address a multiple of 4 */
uint32_t dtc_cmt0_transfer_data[4];

void func(void)
{
    /* Configure the DTC for CMT0 */
    R_DTC_Create(
        PDL_DTC_NORMAL | PDL_DTC_SOURCE_ADDRESS_FIXED | \
        PDL_DTC_DESTINATION_ADDRESS_PLUS | PDL_DTC_SIZE_8 | \
        PDL_DTC_TRIGGER_CMT0,
        dtc_cmt0_transfer_data,
        0x00000A00,
        0x00000B00,
        100,
        0
    );
}
```

### 3) R\_DTC\_Destroy

**Synopsis**

Disable the Data Transfer Controller.

**Prototype**

```
bool R_DTC_Destroy(  
    void    // No parameter is required  
);
```

**Description**

Shutdown the Data Transfer Controller.

**Return value**

True.

**Category**

Data Transfer Controller

**Reference**

R\_DTC\_Control

**Remarks**

- Before calling this function,
  - i. If another peripheral is being used to trigger a DTC transfer, stop the triggers from that peripheral (using Control or Destroy for that peripheral).
  - ii. Use R\_DTC\_Control to stop the DTC.

**Program example**

```
/* RPDL definitions */  
#include "r_pdl_dtc.h"  
  
/* RPDL device-specific definitions */  
#include "r_pdl_definitions.h"  
  
void func(void)  
{  
    /* Shutdown the DTC */  
    R_DTC_Destroy(  
    );  
}
```

## 4) R\_DTC\_Control

### Synopsis

Control the Data Transfer Controller.

### Prototype

```
bool R_DTC_Control (
    uint32_t data1,    // Control options
    uint32_t * data2,  // Transfer data start address
    void * data3,      // Source start address
    void * data4,      // Destination start address
    uint16_t data5,    // Transfer count
    uint8_t data6      // Block size
);
```

### Description

Modify the operation of the Data Transfer Controller.

#### [data1]

Control the operation.

- Stop / Start control

PDL_DTC_STOP or PDL_DTC_START	Enable / re-enable or suspend DTC transfers.
----------------------------------	--

- The transfer registers to be modified, using the selected parameters.

PDL_DTC_UPDATE_SOURCE	The Source Address register, using parameter data3.
PDL_DTC_UPDATE_DESTINATION	The Transfer Address register, using parameter data4.
PDL_DTC_UPDATE_COUNT	The Transfer Count register, using parameter data5.
PDL_DTC_UPDATE_BLOCK_SIZE	The Block Size register, using parameter data6.

- Transfer trigger control

When the transfer count specified in R\_DTC\_Create is completed, the DTC will ignore further interrupts from that trigger source.

If you require the interrupt to trigger another transfer, specify the trigger used in the relevant call of R\_DTC\_Create.

#### [data2]

If transfer registers are to be modified, specify the start address of the transfer data area (the same as that declared in R\_DTC\_Create).

If no registers are to be modified, specify PDL\_NO\_PTR.

#### [data3]

The new source start address. The valid range depends on the address mode (short or full). Specify PDL\_NO\_PTR if not required.

#### [data4]

The new destination start address. The valid range depends on the address mode (short or full). Specify PDL\_NO\_PTR if not required.

#### [data5]

The new number of transfers to take place.

For normal or block mode, valid between 0 and 65535 (0 = 65536 transfers).

For repeat mode, valid between 0 and 255 (0 = 256 transfers).

Specify PDL\_NO\_DATA if not required.

#### [data6]

The new size of each block transfer. Valid between 1 and 255 units.

Ignored in normal or repeat mode.

Specify PDL\_NO\_DATA if not required.

### Return value

True if all parameters are valid and exclusive; otherwise false.

### Category

Data Transfer Controller

### Reference

R\_DTC\_Create

**Remarks**

- This function must be called in order to start the DTC (R\_DTC\_Create must be called at least once before starting the DTC).

**Program example**

```
/* RPDL definitions */
#include "r_pdl_dtc.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Start the controller */
    R_DTC_Control(
        PDL_DTC_START,
        PDL_NO_PTR,
        PDL_NO_PTR,
        PDL_NO_PTR,
        PDL_NO_DATA,
        PDL_NO_DATA
    );

    /* Update the parameters for CMT0-triggered transfers */
    R_DTC_Control(
        PDL_DTC_UPDATE_DESTINATION | PDL_DTC_UPDATE_COUNT,
        dtc_cmt0_transfer_data,
        PDL_NO_PTR,
        0x00000B00,
        100,
        PDL_NO_DATA
    );
}
```



## 5) R\_DTC\_GetStatus

### Synopsis

Check the status of the Data Transfer Controller.

### Prototype

```
bool R_DTC_GetStatus(
    uint32_t * data1, // Transfer data start address
    uint16_t * data2, // Status flags pointer
    uint32_t * data3, // Current source address pointer
    uint32_t * data4, // Current destination address pointer
    uint16_t * data5, // Current transfer count pointer
    uint8_t * data6   // Current block size count pointer
);
```

### Description

Return status flags and current channel registers.

#### [data1]

The start address of the transfer data area.

If all parameters data3, data4, data5 and data6 are not required, specify PDL\_NO\_PTR.

#### [data2]

The status flags shall be stored in the following format.

Specify PDL\_NO\_PTR if the status flags are not required.

b15	b14 – b8	b7 - b0
0: Idle 1: A transfer is in progress	0	The trigger vector (valid only when b15 = 1)

#### [data3]

Where the current source address shall be stored. Ignored if data1 is set to PDL\_NO\_PTR.

If this value is not required, specify PDL\_NO\_PTR.

#### [data4]

Where the current destination address shall be stored. Ignored if data1 is set to PDL\_NO\_PTR.

If this value is not required, specify PDL\_NO\_PTR.

#### [data5]

Where the current transfer count shall be stored. Ignored if data1 is set to PDL\_NO\_PTR.

If this value is not required, specify PDL\_NO\_PTR.

#### [data6]

Where the current block size count shall be stored. Ignored if data1 is set to PDL\_NO\_PTR.

If this value is not required, specify PDL\_NO\_PTR.

### Return value

True if all parameters are valid and exclusive; otherwise false.

### Category

Data Transfer Controller

### Reference

R\_DTC\_Create

### Remarks

- The start address of the transfer data area is the same as that declared in R\_DTC\_Create.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_dtc.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

/* Declared in the R_DTC_Create example */
extern uint32_t dtc_cmt0_transfer_data[];

void func(void)
{
    uint16_t StatusValue;
    uint32_t SourceAddr;

    /* Read the status and current source address for the CMT0 transfer
    */
    R_DTC_GetStatus(
        dtc_cmt0_transfer_data,
        &StatusValue,
        &SourceAddr,
        PDL_NO_PTR,
        PDL_NO_PTR,
        PDL_NO_PTR
    );
}
```

## 4.2.10. Multi-Function Timer Pulse Unit

## 1) R\_MTU3\_Set

**Synopsis**

Configure the Multi-function Timer Pulse Unit.

**Prototype**

```
bool R_MTU3_Set(
    uint8_t data    // Configuration
);
```

**Description**

Set up the global MTU options.

**[data]**

Configure the global options. Use "|" to separate each selection.

## • Pin selection

PDL_MTU3_PIN_0A_A or PDL_MTU3_PIN_0A_B	Select the -A or -B pin for MTIOC0A.
PDL_MTU3_PIN_0B_A or PDL_MTU3_PIN_0B_B	Select the -A or -B pin for MTIOC0B.
PDL_MTU3_PIN_CLKABCD_A or PDL_MTU3_PIN_CLKABCD_B or PDL_MTU3_PIN_CLKABCD_C	Select the -A, -B or -C pins for MTCLKA, MTCLKB, MTCLKD and MTCLKD.

**Return value**

True if all parameters are valid and exclusive; otherwise false.

**Category**

Multi-function Timer Pulse Unit

**Reference**

R\_MTU3\_Create

**Remarks**

- Before calling R\_MTU3\_Create, call this function to configure the relevant pins.
- There are no -C pin options for signals MTCLKA or MTCLKB.
- The 64-pin device package does not support all of the MTCLK pin options.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_mtu3.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Configure the MTU pins */
    R_MTU3_Set(
        PDL_MTU3_PIN_0A_A | PDL_MTU3_PIN_0B_A | PDL_MTU3_PIN_CLKABCD_B
    );
}
```

## 2) R\_MTU3\_Create

### Synopsis

Configure an MTU channel.

### Prototype

```
bool R_MTU3_Create(
    uint8_t data1,           // Channel selection
    R_MTU3_Create_structure ptr // A pointer to the structure
);
```

R\_MTU3\_Create\_structure members:

```
uint32_t data2 // Configuration selection
uint32_t data3 // Configuration selection
uint32_t data4 // Configuration selection
uint32_t data5 // Configuration selection
uint32_t data6 // Configuration selection
uint32_t data7 // Configuration selection
uint32_t data8 // Configuration selection
uint16_t data9 // Register value
uint16_t data10 // Register value
uint16_t data11 // Register value
uint16_t data12 // Register value
uint16_t data13 // Register value
uint16_t data14 // Register value
uint16_t data15 // Register value
uint16_t data16 // Register value
uint16_t data17 // Register value
uint16_t data18 // Register value
uint16_t data19 // Register value
void * func1 // Callback function
void * func2 // Callback function
void * func3 // Callback function
void * func4 // Callback function
uint8_t data20 // Interrupt priority level
void * func5 // Callback function
void * func6 // Callback function
void * func7 // Callback function
void * func8 // Callback function
uint8_t data21 // Interrupt priority level
```

### Description (1/10)

Set up a 16-bit MTU channel.

#### [data1]

The channel number *n* (where *n* = 0 to 7).

#### [data2]

Configure the channel mode.

If multiple selections are required, use “|” to separate each selection.

The default settings are shown in **bold**. Specify PDL\_NO\_DATA to use the defaults.

- Operation mode. Valid for *n* ≠ 5, unless stated otherwise.

PDL_MTU3_MODE_NORMAL or	Normal operation.
PDL_MTU3_MODE_PWM1 or	Pulse Width Modulation (PWM) mode 1.
PDL_MTU3_MODE_PWM2 or	Pulse Width Modulation (PWM) mode 2. Valid for <i>n</i> = 0, 1 and 2.
PDL_MTU3_MODE_PHASE1 or PDL_MTU3_MODE_PHASE2 or PDL_MTU3_MODE_PHASE3 or PDL_MTU3_MODE_PHASE4 or	Phase counting mode 1, 2, 3 or 4. Valid for <i>n</i> = 1 and 2.
PDL_MTU3_MODE_PWM_RS or	Reset-synchronised PWM mode. Valid for <i>n</i> = 3 and 6.
PDL_MTU3_MODE_PWM_COMP1 or PDL_MTU3_MODE_PWM_COMP2 or PDL_MTU3_MODE_PWM_COMP3	Complementary PWM mode 1, 2 or 3. Valid for <i>n</i> = 3 and 6.

**Description (2/10)**

- Synchronous mode. Valid for  $n \neq 5$ .

<b>PDL_MTU3_SYNC_DISABLE</b> or <b>PDL_MTU3_SYNC_ENABLE</b>	Disable or enable synchronous presetting / clearing.
--	--

- DTC event trigger control. Valid for  $n \neq 5$ , unless stated otherwise.

<b>PDL_MTU3_TGRA_DTC_TRIGGER_DISABLE</b> or <b>PDL_MTU3_TGRA_DTC_TRIGGER_ENABLE</b>	TGRA compare match or input capture.
<b>PDL_MTU3_TGRB_DTC_TRIGGER_DISABLE</b> or <b>PDL_MTU3_TGRB_DTC_TRIGGER_ENABLE</b>	TGRB compare match or input capture.
<b>PDL_MTU3_TGRC_DTC_TRIGGER_DISABLE</b> or <b>PDL_MTU3_TGRC_DTC_TRIGGER_ENABLE</b>	TGRC compare match or input capture. Valid for $n = 0, 3, 4, 6$ and $7$ .
<b>PDL_MTU3_TGRD_DTC_TRIGGER_DISABLE</b> or <b>PDL_MTU3_TGRD_DTC_TRIGGER_ENABLE</b>	TGRD compare match or input capture. Valid for $n = 0, 3, 4, 6$ and $7$ .
<b>PDL_MTU3_TCIV_DTC_TRIGGER_DISABLE</b> or <b>PDL_MTU3_TCIV_DTC_TRIGGER_ENABLE</b>	Counter overflow or underflow. Valid for $n = 4$ and $7$ .

- DTC event trigger control. Valid for  $n = 5$ .

<b>PDL_MTU3_TGRU_DTC_TRIGGER_DISABLE</b> or <b>PDL_MTU3_TGRU_DTC_TRIGGER_ENABLE</b>	TGRU compare match or input capture.
<b>PDL_MTU3_TGRV_DTC_TRIGGER_DISABLE</b> or <b>PDL_MTU3_TGRV_DTC_TRIGGER_ENABLE</b>	TGRV compare match or input capture.
<b>PDL_MTU3_TGRW_DTC_TRIGGER_DISABLE</b> or <b>PDL_MTU3_TGRW_DTC_TRIGGER_ENABLE</b>	TGRW compare match or input capture.

**Description (3/10)****[data3]**

Configure the counter operation.

If multiple selections are required, use “|” to separate each selection.

The default settings are shown in **bold**. Specify PDL\_NO\_DATA to use the defaults.

- TCNT counter clock source selection. Valid for  $n \neq 5$ , unless stated otherwise.

<b>PDL_MTU3_CLK_ICLK_DIV_1</b> or PDL_MTU3_CLK_ICLK_DIV_4 or PDL_MTU3_CLK_ICLK_DIV_16 or PDL_MTU3_CLK_ICLK_DIV_64 or	The internal clock signal $ICLK \div 1, 4, 16$ or $64$ .
PDL_MTU3_CLK_ICLK_DIV_256 or	$ICLK \div 256$ . Valid for $n = 1, 3, 4, 6$ and $7$ .
PDL_MTU3_CLK_ICLK_DIV_1024 or	$ICLK \div 1024$ . Valid for $n = 2, 3, 4, 6$ and $7$ .
PDL_MTU3_CLK_MTCLKA or	MTCLKA pin input. Valid for $n = 0$ to $4$ .
PDL_MTU3_CLK_MTCLKB or	MTCLKB pin input. Valid for $n = 0$ to $4$ .
PDL_MTU3_CLK_MTCLKC or	MTCLKC pin input. Valid for $n = 0$ and $2$ .
PDL_MTU3_CLK_MTCLKD or	MTCLKD pin input. Valid for $n = 0$ .
PDL_MTU3_CLK_CASCADE	The overflow / underflow signal from MTU channel 2. Valid for $n = 1$ .

- TCNT counter clock edge selection. Valid for  $n \neq 5$ .

<b>PDL_MTU3_CLK_RISING</b> or PDL_MTU3_CLK_FALLING or PDL_MTU3_CLK_BOTH	The TCNT counter clock signal shall be counted on rising, falling or both edges.
---	--

- TCNT counter clearing. Valid for  $n \neq 5$ , unless stated otherwise.

<b>PDL_MTU3_CLEAR_DISABLE</b> or	Clearing is disabled.
PDL_MTU3_CLEAR_TGRA or	Cleared by TGRA compare match or input capture.
PDL_MTU3_CLEAR_TGRB or	Cleared by TGRB compare match or input capture.
PDL_MTU3_CLEAR_SYNC or	Cleared by counter clearing on another channel configured for synchronous operation.
PDL_MTU3_CLEAR_TGRC or	Cleared by TGRC compare match or input capture. Valid for $n = 0, 3, 4, 6$ and $7$ .
PDL_MTU3_CLEAR_TGRD	Cleared by TGRD compare match or input capture. Valid for $n = 0, 3, 4, 6$ and $7$ .

- Counter clock source selection. Valid for  $n = 5$ .

<b>PDL_MTU3_CLKU_ICLK_DIV_1</b> or PDL_MTU3_CLKU_ICLK_DIV_4 or PDL_MTU3_CLKU_ICLK_DIV_16 or PDL_MTU3_CLKU_ICLK_DIV_64 or	Counter TCNTU is supplied by the internal clock signal $ICLK \div 1, 4, 16$ or $64$ .
<b>PDL_MTU3_CLKV_ICLK_DIV_1</b> or PDL_MTU3_CLKV_ICLK_DIV_4 or PDL_MTU3_CLKV_ICLK_DIV_16 or PDL_MTU3_CLKV_ICLK_DIV_64 or	Counter TCNTV is supplied by the internal clock signal $ICLK \div 1, 4, 16$ or $64$ .
<b>PDL_MTU3_CLKW_ICLK_DIV_1</b> or PDL_MTU3_CLKW_ICLK_DIV_4 or PDL_MTU3_CLKW_ICLK_DIV_16 or PDL_MTU3_CLKW_ICLK_DIV_64 or	Counter TCNTW is supplied by the internal clock signal $ICLK \div 1, 4, 16$ or $64$ .

- Counter clearing (U, V and W counters). Valid for  $n = 5$ .

<b>PDL_MTU3_CLEAR_TGRU_DISABLE</b> or PDL_MTU3_CLEAR_TGRU_ENABLE	Disable or enable clearing of TCNTU by TGRU compare match or input capture.
<b>PDL_MTU3_CLEAR_TGRV_DISABLE</b> or PDL_MTU3_CLEAR_TGRV_ENABLE	Disable or enable clearing of TCNTV by TGRV compare match or input capture.
<b>PDL_MTU3_CLEAR_TGRW_DISABLE</b> or PDL_MTU3_CLEAR_TGRW_ENABLE	Disable or enable clearing of TCNTW by TGRW compare match or input capture.

**Description (4/10)****[data4]**

Configure the ADC trigger operation.

If multiple selections are required, use “|” to separate each selection.

The default settings are shown in **bold**. Specify PDL\_NO\_DATA to use the defaults.

- ADC conversion trigger control. Valid for  $n \neq 5$ , unless stated otherwise.

<b>PDL_MTU3_ADC_TRIG_TGRA_DISABLE</b> or PDL_MTU3_ADC_TRIG_TGRA_ENABLE	Disable or enable ADC start requests on a TGRA compare match or input capture.
<b>PDL_MTU3_ADC_TRIG_TGRE_DISABLE</b> or PDL_MTU3_ADC_TRIG_TGRE_ENABLE	Disable or enable ADC start requests on a TGRE compare match or input capture. Valid only for $n = 0$ .
<b>PDL_MTU3_ADC_TRIG_TROUGH_DISABLE</b> or PDL_MTU3_ADC_TRIG_TROUGH_ENABLE	Disable or enable ADC start requests on a TCNT underflow. Valid for $n = 4$ and $7$ in complementary PWM mode.

- Control ADC trigger interrupt skipping. Valid for  $n = 4$  and  $7$  in complementary PWM mode.

<b>PDL_MTU3_ADC_TRIG_A_TROUGH_INT_SKIP_DISABLE</b> or PDL_MTU3_ADC_TRIG_A_TROUGH_INT_SKIP_ENABLE	Disable or link interrupt skipping to ADC trigger TRGnAN on a TCNT underflow.
<b>PDL_MTU3_ADC_TRIG_B_TROUGH_INT_SKIP_DISABLE</b> or PDL_MTU3_ADC_TRIG_B_TROUGH_INT_SKIP_ENABLE	Disable or link interrupt skipping to ADC trigger TRGnBN on a TCNT underflow.
<b>PDL_MTU3_ADC_TRIG_A_CREST_INT_SKIP_DISABLE</b> or PDL_MTU3_ADC_TRIG_A_CREST_INT_SKIP_ENABLE	Disable or link interrupt skipping to ADC trigger TRGnAN on a TGRA compare match.
<b>PDL_MTU3_ADC_TRIG_B_CREST_INT_SKIP_DISABLE</b> or PDL_MTU3_ADC_TRIG_B_CREST_INT_SKIP_ENABLE	Disable or link interrupt skipping to ADC trigger TRGnBN on a TGRA compare match.

- Control ADC triggers. Valid for  $n = 4$  and  $7$  in complementary PWM mode unless stated otherwise.

<b>PDL_MTU3_ADC_TRIG_A_DOWN_DISABLE</b> or PDL_MTU3_ADC_TRIG_A_DOWN_ENABLE	Disable or enable ADC trigger TRGnAN requests during down-count operation.
<b>PDL_MTU3_ADC_TRIG_B_DOWN_DISABLE</b> or PDL_MTU3_ADC_TRIG_B_DOWN_ENABLE	Disable or enable ADC trigger TRGnBN requests during down-count operation.
<b>PDL_MTU3_ADC_TRIG_A_UP_DISABLE</b> or PDL_MTU3_ADC_TRIG_A_UP_ENABLE	Disable or enable ADC trigger TRGnAN requests during up-count operation. This option can be selected in other modes.
<b>PDL_MTU3_ADC_TRIG_B_UP_DISABLE</b> or PDL_MTU3_ADC_TRIG_B_UP_ENABLE	Disable or enable ADC trigger TRGnBN requests during up-count operation. This option can be selected in other modes.

**Description (5/10)****[data5]**

Configure the buffer operation.

If multiple selections are required, use “|” to separate each selection.

The default settings are shown in **bold**. Specify PDL\_NO\_DATA to use the defaults.

- Double buffer control. Valid for n = 3 and 6 in complementary PWM mode 3.

<b>PDL_MTU3_BUFFER_DOUBLE_DISABLE</b> or <b>PDL_MTU3_BUFFER_DOUBLE_ENABLE</b>	Disable or enable the double buffer function.
--	---

- Control the cycle set buffer transfer timing. Valid for n = 4 and 7.

<b>PDL_MTU3_CSB_DISABLE</b> or <b>PDL_MTU3_CSB_CREST</b> or <b>PDL_MTU3_CSB_TROUGH</b> or <b>PDL_MTU3_CSB_BOTH</b>	Select no transfer, transfer on crest detection, transfer on trough detection or transfer on crest and trough detection.
---	---

- Buffer operation

<b>PDL_MTU3_BUFFER_AC_DISABLE</b> or <b>PDL_MTU3_BUFFER_AC_ENABLE</b>	Disable or enable buffer operation for registers TGRA and TGRC. Valid for n = 0, 3, 4, 6 and 7.
<b>PDL_MTU3_BUFFER_BD_DISABLE</b> or <b>PDL_MTU3_BUFFER_BD_ENABLE</b>	Disable or enable buffer operation for registers TGRB and TGRD. Valid for n = 0, 3, 4, 6 and 7.
<b>PDL_MTU3_BUFFER_EF_DISABLE</b> or <b>PDL_MTU3_BUFFER_EF_ENABLE</b>	Disable or enable buffer operation for registers TGRE and TGRF. Valid for n = 0.

- Buffer data transfer

<b>PDL_MTU3_BUFFER_AC_CM_A</b> or <b>PDL_MTU3_BUFFER_AC_TCNT_CLR</b>	Transfer the data from TGRC to TGRA when a compare match A occurs or when TCNT is cleared in each channel. Valid for n = 0, 3, 4, 6 and 7.
<b>PDL_MTU3_BUFFER_BD_CM_B</b> or <b>PDL_MTU3_BUFFER_BD_TCNT_CLR</b>	Transfer the data from TGRD to TGRB when a compare match B occurs or when TCNT is cleared in each channel. Valid for n = 0, 3, 4, 6 and 7.
<b>PDL_MTU3_BUFFER_EF_CM_E</b> or <b>PDL_MTU3_BUFFER_EF_TCNT_CLR</b>	Transfer the data from TGRF to TGRE when a compare match E occurs or when TCNT is cleared in either channel. Valid for n = 0.



**Description (6/10)****[data6]**

Configure the operation for general registers TGRA and TGRB. Valid for  $n \neq 5$ .  
 If multiple selections are required, use “|” to separate each selection.  
 The default settings are shown in **bold**. Specify PDL\_NO\_DATA to use the defaults.

- Input capture / output compare control for register TGRA

<b>PDL_MTU3_A_OC_DISABLED</b> or PDL_MTU3_A_OC_LOW or PDL_MTU3_A_OC_LOW_CM_HIGH or  PDL_MTU3_A_OC_LOW_CM_INV or  PDL_MTU3_A_OC_HIGH_CM_LOW or  PDL_MTU3_A_OC_HIGH or PDL_MTU3_A_OC_HIGH_CM_INV or	MTIOCnA output disabled. MTIOCnA output low. MTIOCnA initial output low; goes high at compare match. MTIOCnA initial output low; toggles at compare match. MTIOCnA initial output high; goes low at compare match. MTIOCnA output high. MTIOCnA initial output high; toggles at compare match.
PDL_MTU3_A_IC_RISING_EDGE or PDL_MTU3_A_IC_FALLING_EDGE or PDL_MTU3_A_IC_BOTH_EDGES or	Input capture at MTIOCnA rising edge. Input capture at MTIOCnA falling edge. Input capture at MTIOCnA both edges.
PDL_MTU3_A_IC_COUNT or	Input capture at channel (n+1) up-count or down-count. Valid only for $n = 0$ .
PDL_MTU3_A_IC_CM_IC	Input capture at channel (n-1) TGRA compare match or input capture. Valid only for $n = 1$ .

- Input capture / output compare control for register TGRB.

<b>PDL_MTU3_B_OC_DISABLED</b> or PDL_MTU3_B_OC_LOW or PDL_MTU3_B_OC_LOW_CM_HIGH or  PDL_MTU3_B_OC_LOW_CM_INV or  PDL_MTU3_B_OC_HIGH_CM_LOW or  PDL_MTU3_B_OC_HIGH or PDL_MTU3_B_OC_HIGH_CM_INV or	MTIOCnB output disabled. MTIOCnB output low. MTIOCnB initial output low; goes high at compare match. MTIOCnB initial output low; toggles at compare match. MTIOCnB initial output high; goes low at compare match. MTIOCnB output high. MTIOCnB initial output high; toggles at compare match.
PDL_MTU3_B_IC_RISING_EDGE or PDL_MTU3_B_IC_FALLING_EDGE or PDL_MTU3_B_IC_BOTH_EDGES or	Input capture at MTIOCnB rising edge. Input capture at MTIOCnB falling edge. Input capture at MTIOCnB both edges.
PDL_MTU3_B_IC_COUNT or	Input capture at channel (n+1) up-count or down-count. Valid only for $n = 0$ .
PDL_MTU3_B_IC_CM_IC	Input capture at channel (n-1) TGRB compare match or input capture. Valid only for $n = 1$ .

- Cascade input capture control. Valid in cascade mode for  $n = 1$ .  
 Channel 1 forms the higher 16 bits and channel 2 forms the lower 16 bits.

<b>PDL_MTU3_CASCADE_AL_IC_EXC_H</b> or PDL_MTU3_CASCADE_AL_IC_INC_H	Exclude or include pin MTIOC1A in the TGRA input capture conditions for channel 2.
<b>PDL_MTU3_CASCADE_BL_IC_EXC_H</b> or PDL_MTU3_CASCADE_BL_IC_INC_H	Exclude or include pin MTIOC1B in the TGRB input capture conditions for channel 2.
<b>PDL_MTU3_CASCADE_AH_IC_EXC_L</b> or PDL_MTU3_CASCADE_AH_IC_INC_L	Exclude or include pin MTIOC2A in the TGRA input capture conditions for channel 1.
<b>PDL_MTU3_CASCADE_BH_IC_EXC_L</b> or PDL_MTU3_CASCADE_BH_IC_INC_L	Exclude or include pin MTIOC2B in the TGRB input capture conditions for channel 1.

**Description (7/10)****[data7]**

Configure the operation for general registers TGRC and TGRD. Valid for n = 0, 3, 4, 6 and 7.  
If multiple selections are required, use “|” to separate each selection.  
The default settings are shown in **bold**. Specify PDL\_NO\_DATA to use the defaults.

- Input capture / output compare control for register TGRC.

<b>PDL_MTU3_C_OC_DISABLED</b> or PDL_MTU3_C_OC_LOW or PDL_MTU3_C_OC_LOW_CM_HIGH or  PDL_MTU3_C_OC_LOW_CM_INV or  PDL_MTU3_C_OC_HIGH_CM_LOW or  PDL_MTU3_C_OC_HIGH or PDL_MTU3_C_OC_HIGH_CM_INV or	MTIOcNc output disabled. MTIOcNc output low. MTIOcNc initial output low; goes high at compare match. MTIOcNc initial output low; toggles at compare match. MTIOcNc initial output high; goes low at compare match. MTIOcNc output high. MTIOcNc initial output high; toggles at compare match.
PDL_MTU3_C_IC_RISING_EDGE or PDL_MTU3_C_IC_FALLING_EDGE or PDL_MTU3_C_IC_BOTH_EDGES or	Input capture at MTIOcNc rising edge. Input capture at MTIOcNc falling edge. Input capture at MTIOcNc both edges.
PDL_MTU3_C_IC_COUNT	Input capture at channel (n+1) up-count or down-count. Valid only for n = 0.

- Input capture / output compare control for register TGRD.

<b>PDL_MTU3_D_OC_DISABLED</b> or PDL_MTU3_D_OC_LOW or PDL_MTU3_D_OC_LOW_CM_HIGH or  PDL_MTU3_D_OC_LOW_CM_INV or  PDL_MTU3_D_OC_HIGH_CM_LOW or  PDL_MTU3_D_OC_HIGH or PDL_MTU3_D_OC_HIGH_CM_INV or	MTIOcNd output disabled. MTIOcNd output low. MTIOcNd initial output low; goes high at compare match. MTIOcNd initial output low; toggles at compare match. MTIOcNd initial output high; goes low at compare match. MTIOcNd output high. MTIOcNd initial output high; toggles at compare match.
PDL_MTU3_D_IC_RISING_EDGE or PDL_MTU3_D_IC_FALLING_EDGE or PDL_MTU3_D_IC_BOTH_EDGES or	Input capture at MTIOcNd rising edge. Input capture at MTIOcNd falling edge. Input capture at MTIOcNd both edges.
PDL_MTU3_D_IC_COUNT	Input capture at channel (n+1) up-count or down-count. Valid only for n = 0.

**Description (8/10)****[data8]**

Configure the input capture / compare match control for general registers TGRU, TRGV and TGRW. Valid for n = 5.

The default settings are shown in **bold**. Specify PDL\_NO\_DATA to use the defaults.

- Input capture / compare match control for register TGRU.

<b>PDL_MTU3_U_CM</b> or	Compare match.
PDL_MTU3_U_IC_RISING_EDGE or	Input capture at MTICnU rising edge.
PDL_MTU3_U_IC_FALLING_EDGE or	Input capture at MTICnU falling edge.
PDL_MTU3_U_IC_BOTH_EDGES or	Input capture at MTICnU both edges.
PDL_MTU3_U_IC_PWM_LOW_TROUGH or	Input capture at trough,
PDL_MTU3_U_IC_PWM_LOW_CREST or	crest or
PDL_MTU3_U_IC_PWM_LOW_BOTH or	both for low pulse width measurement.
PDL_MTU3_U_IC_PWM_HIGH_TROUGH or	Input capture at trough,
PDL_MTU3_U_IC_PWM_HIGH_CREST or	crest or
PDL_MTU3_U_IC_PWM_HIGH_BOTH	both for high pulse width measurement.

- Input capture / compare match control for register TRGV.

<b>PDL_MTU3_V_CM</b> or	Compare match.
PDL_MTU3_V_IC_RISING_EDGE or	Input capture at MTICnV rising edge.
PDL_MTU3_V_IC_FALLING_EDGE or	Input capture at MTICnV falling edge.
PDL_MTU3_V_IC_BOTH_EDGES or	Input capture at MTICnV both edges.
PDL_MTU3_V_IC_PWM_LOW_TROUGH or	Input capture at trough,
PDL_MTU3_V_IC_PWM_LOW_CREST or	crest or
PDL_MTU3_V_IC_PWM_LOW_BOTH or	both for low pulse width measurement.
PDL_MTU3_V_IC_PWM_HIGH_TROUGH or	Input capture at trough,
PDL_MTU3_V_IC_PWM_HIGH_CREST or	crest or
PDL_MTU3_V_IC_PWM_HIGH_BOTH	both for high pulse width measurement.

- Input capture / compare match control for register TGRW.

<b>PDL_MTU3_W_CM</b> or	Compare match.
PDL_MTU3_W_IC_RISING_EDGE or	Input capture at MTICnW rising edge.
PDL_MTU3_W_IC_FALLING_EDGE or	Input capture at MTICnW falling edge.
PDL_MTU3_W_IC_BOTH_EDGES or	Input capture at MTICnW both edges.
PDL_MTU3_W_IC_PWM_LOW_TROUGH or	Input capture at trough,
PDL_MTU3_W_IC_PWM_LOW_CREST or	crest or
PDL_MTU3_W_IC_PWM_LOW_BOTH or	both for low pulse width measurement.
PDL_MTU3_W_IC_PWM_HIGH_TROUGH or	Input capture at trough,
PDL_MTU3_W_IC_PWM_HIGH_CREST or	crest or
PDL_MTU3_W_IC_PWM_HIGH_BOTH	both for high pulse width measurement.

**[data9]**

For n ≠ 5: The timer counter TCNT value.

For n = 5: The timer counter TCNTU value.

**[data10]**

For n ≠ 5: The register TGRA value.

For n = 5: The timer counter TCNTV value.

**[data11]**

For n ≠ 5: The register TGRB value.

For n = 5: The timer counter TCNTW value.

**[data12]**

For n = 0, 3, 4, 6 and 7: The register TGRC value.

For n = 5: The register TGRU value.

Ignored for other channels.

**[data13]**

For n = 0, 3, 4, 6 and 7: The register TGRD value.

For n = 5: The register TRGV value.

Ignored for other channels.

**Description (9/10)****[data14]**

For n = 0, 3, 4, 6 and 7: The register TGRE value.  
 For n = 5: The register TGRW value.  
 Ignored for other channels.

**[data15]**

For n = 0, 4 and 7: The register TGRF value. Ignored for other channels.

**[data16]**

For n = 4 and 7: The register TADCORA value. Ignored for other channels.

**[data17]**

For n = 4 and 7: The register TADCORB value. Ignored for other channels.

**[data18]**

For n = 4 and 7: The register TADCOBRA value. Ignored for other channels.

**[data19]**

For n = 4 and 7: The register TADCOBRB value. Ignored for other channels.

**[func1]**

For n ≠ 5: The function to be called when a TGRA event occurs.  
 For n = 5: The function to be called when a TGRU event occurs.  
 Specify PDL\_NO\_FUNC if not required.

**[func2]**

For n ≠ 5: The function to be called when a TGRB event occurs.  
 For n = 5: The function to be called when a TGRV event occurs.  
 Specify PDL\_NO\_FUNC if not required.

**[func3]**

For n = 0, 3, 4, 6 and 7: The function to be called when a TGRC event occurs.  
 For n = 5: The function to be called when a TGRW event occurs.  
 Specify PDL\_NO\_FUNC if not required.

**[func4]**

For n = 0, 3, 4, 6 and 7: The function to be called when a TGRD event occurs.  
 Specify PDL\_NO\_FUNC if not required.

**[data20]**

The interrupt priority level for TGR(A to D or U to W) events.  
 Select between 1 (lowest priority) and 15 (highest priority).  
 This parameter will be ignored if PDL\_NO\_FUNC is specified for all parameters func(1 to 4).

**[func5]**

For n = 0: The function to be called when a TGRE event occurs.  
 Specify PDL\_NO\_FUNC if not required.

**[func6]**

For n = 0: The function to be called when a TGRF event occurs.  
 Specify PDL\_NO\_FUNC if not required.

**[func7]**

For n = 0 to 3 or 6 to 7: The function to be called when an overflow occurs.  
 For n = 4: The function to be called when an overflow or underflow occurs.  
 Specify PDL\_NO\_FUNC if not required.

**[func8]**

For n = 1 and 2: The function to be called when an underflow occurs.  
 Specify PDL\_NO\_FUNC if not required.

<b>Description (10/10)</b>	<p><b>[data21]</b>  The interrupt priority level for TGRE, TGRF, overflow or underflow events.  Select between 1 (lowest priority) and 15 (highest priority).  This parameter will be ignored if PDL_NO_FUNC is specified for all parameters func(5 to 8).</p>
<b>Return value</b>	True if all parameters are valid and exclusive; otherwise false.
<b>Category</b>	Multi-function Timer Pulse Unit
<b>Reference</b>	R_MTU3_Set, R_MTU3_ControlChannel, R_MTU3_ControlUnit
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• If an external clock input pin (MTCLKx) or I/O pin (MTIOCnx) is made active, this function will configure that pin for input or output and disable other functions on that pin.</li> <li>• The alternative pins are assigned using function R_MTU3_Set.</li> <li>• Either R_MTU3_ControlChannel or R_MTU3_ControlUnit must be used to start the timers.</li> <li>• If a callback function is specified, this function will enable the relevant CPU interrupt. Please see the notes on callback function usage in §6.</li> <li>• A callback function is executed by the interrupt processing function. This means that no other interrupt can be processed until the callback function has completed.</li> <li>• If the channel is configured for phase counting mode, the counter clock source setting is ignored.</li> <li>• If reset-synchronised or complementary PWM mode is selected, the associated channel (4 or 7) will be set to normal mode.</li> <li>• If buffer operation is selected for registers TGRA and TGRC, input capture / output compare is not valid for register TGRC.</li> <li>• If buffer operation is selected for registers TGRB and TGRD, input capture / output compare is not valid for register TGRD.</li> <li>• If synchronous mode is required, at least two channels must be enabled for synchronous operation.</li> <li>• A companion function, R_MTU3_Create_load_defaults, can be used to load the default values into the structure.</li> </ul>

**Program example**

```

/* RPDL definitions */
#include "r_pdl_mtu3.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Allocate a copy of the structure for the selected channel */
    R_MTU3_Create_structure ch4_parameters;

    /* Load the defaults */
    R_MTU3_Create_load_defaults(&ch4_parameters);

    /* Set the non-default options for channel 4 */
    ch4_parameters.data2 = PDL_MTU3_MODE_NORMAL | PDL_MTU3_SYNC_ENABLE |
PDL_MTU3_TGRA_DTC_TRIGGER_ENABLE;
    ch4_parameters.data3 = PDL_MTU3_CLK_ICLK_DIV_4;
    ch4_parameters.data5 = PDL_MTU3_BUFFER_AC_CM_A;
    ch4_parameters.data7 = PDL_MTU3_C_OC_HIGH_CM_LOW;
    ch4_parameters.data9 = 0;
    ch4_parameters.data10 = 199;
    ch4_parameters.data11 = 99;
    ch4_parameters.data12 = 50;
    ch4_parameters.data13 = 100;
    ch4_parameters.data14 = 0;
    ch4_parameters.data15 = 0;

    R_MTU3_Create(
        4,
        &ch4_parameters
    );
}

```

### 3) R\_MTU3\_Destroy

**Synopsis**

Disable the Multi-function Timer Pulse Unit.

**Prototype**

```
bool R_MTU3_Destroy(  
    void    // No parameter is required  
);
```

**Description**

Shut down a timer pulse unit

**Return value**

True.

**Category**

Multi-function Timer Pulse Unit

**Reference**

None.

**Remarks**

- The unit is put into the stop state to reduce power consumption.

**Program example**

```
/* RPDL definitions */  
#include "r_pdl_mtu3.h"  
  
/* RPDL device-specific definitions */  
#include "r_pdl_definitions.h"  
  
void func(void)  
{  
    /* Shutdown all MTU channels */  
    R_MTU3_Destroy(  
        );  
}
```

#### 4) R\_MTU3\_ControlChannel

##### Synopsis

Control an MTU channel.

##### Prototype

```
bool R_MTU3_ControlChannel(
    uint8_t data1,                // Channel selection
    R_MTU3_ControlChannel_structure ptr // A pointer to the structure
);
```

R\_MTU3\_ControlChannel\_structure members:

```
uint8_t data2    // Control settings
uint16_t data3   // Register selection
uint16_t data4   // Register value
uint16_t data5   // Register value
uint16_t data6   // Register value
uint16_t data7   // Register value
uint16_t data8   // Register value
uint16_t data9   // Register value
uint16_t data10  // Register value
uint16_t data11  // Register value
uint16_t data12  // Register value
```

##### Description (1/2)

Modify a timer channel's registers.

##### [data1]

The channel number *n* (where *n* = 0 to 7).

##### [data2]

The channel settings to be modified.

If multiple selections are required, use “|” to separate each selection.

Specify PDL\_NO\_DATA if no change is required.

- Counter stop / start. Valid for *n* ≠ 5.

PDL_MTU3_STOP	Stop the count operation.
PDL_MTU3_START	Start the count operation.

- Counter stop / Start. Valid for *n* = 5.

PDL_MTU3_STOP_U	Stop the count operation.
PDL_MTU3_STOP_V	
PDL_MTU3_STOP_W	
PDL_MTU3_START_U	Start the count operation.
PDL_MTU3_START_V	
PDL_MTU3_START_W	

##### [data3]

The channel registers to be modified.

If multiple selections are required, use “|” to separate each selection.

Specify PDL\_NO\_DATA if no register change is required.

- The registers to be modified.

For *n* ≠ 5.

PDL_MTU3_REGISTER_COUNTER	Timer counter register (TCNT).
PDL_MTU3_REGISTER_TGRA	General register A.
PDL_MTU3_REGISTER_TGRB	General register B.
PDL_MTU3_REGISTER_TGRC	General register C. Valid for <i>n</i> = 0, 3, 4, 6 and 7.
PDL_MTU3_REGISTER_TGRD	General register D. Valid for <i>n</i> = 0, 3, 4, 6 and 7.
PDL_MTU3_REGISTER_TGRE	General register E. Valid for <i>n</i> = 0, 3, 4, 6 and 7.
PDL_MTU3_REGISTER_TGRF	General register F. Valid for <i>n</i> = 0, 4 and 7.
PDL_MTU3_REGISTER_TADCOBRA	ADC start request cycle set buffer A. Valid for <i>n</i> = 4 and 7.
PDL_MTU3_REGISTER_TADCOBRB	ADC start request cycle set buffer B. Valid for <i>n</i> = 4 and 7.

**Description (2/2)**

For n = 5.

PDL_MTU3_REGISTER_COUNTER_U	Timer counter U register (TCNTU).
PDL_MTU3_REGISTER_COUNTER_V	Timer counter V register (TCNTV).
PDL_MTU3_REGISTER_COUNTER_W	Timer counter W register (TCNTW).
PDL_MTU3_REGISTER_TGRU	General register U.
PDL_MTU3_REGISTER_TGRV	General register V.
PDL_MTU3_REGISTER_TGRW	General register W.

**[data4]**

For n ≠ 5: The timer counter TCNT value.

For n = 5: The timer counter TCNTU value.

This will be ignored if the register is not selected.

**[data5]**

For n ≠ 5: The register TGRA value.

For n = 5: The timer counter TCNTV value.

This will be ignored if the register is not selected.

**[data6]**

For n ≠ 5: The register TGRB value.

For n = 5: The timer counter TCNTW value.

This will be ignored if the register is not selected.

**[data7]**

For n = 0, 3, 4, 6 and 7: The register TGRC value.

For n = 5: The register TGRU value.

This will be ignored if the register is not selected.

**[data8]**

For n = 0, 3, 4, 6 and 7: The register TGRD value.

For n = 5: The register TGRV value.

This will be ignored if the register is not selected.

**[data9]**

For n = 0, 3, 4, 6 and 7: The register TGRE value.

For n = 5: The register TGRW value.

This will be ignored if the register is not selected.

**[data10]**

For n = 0, 4 and 7: The general register TGRF value.

This will be ignored if the register is not selected.

**[data11]**

For n = 4 and 7: ADC start request cycle set buffer A.

This will be ignored if the register is not selected.

**[data12]**

For n = 4 and 7: ADC start request cycle set buffer B.

This will be ignored if the register is not selected.

**Return value**

True if the channel number is valid; otherwise false.

**Category**

Multi-function Timer Pulse Unit

**Reference**

R\_MTU3\_Create, R\_MTU3\_ControlUnit

**Remarks**

- Before calling this function, use R\_MTU3\_Create to configure the channel operation.
- Either this function or R\_MTU3\_ControlUnit must be used to start the timers.
- The Stop operation is executed at the start of this function.  
The Start operation is executed at the end.  
Therefore, both options can be selected together with other changes in one function call.



**Program example**

```
/* RPDL definitions */
#include "r_pdl_mtu3.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Allocate a copy of the structure for the selected channel */
    R_MTU3_ControlChannel_structure ch3_parameters;

    /* Set the control options for channel 3 */
    ch3_parameters.data2 = PDL_MTU3_START;
    ch3_parameters.data3 = PDL_MTU3_REGISTER_COUNTER | \
                          PDL_MTU3_REGISTER_TGRB;
    ch3_parameters.data4 = 0xFFDD;
    ch3_parameters.data6 = 0x0020;

    /* Modify the operation of channel 3 */
    R_MTU3_ControlChannel(
        3,
        &ch3_parameters
    );
}
```

## 5) R\_MTU3\_ControlUnit

### Synopsis

Control the Multi-function Timer Pulse Unit.

### Prototype

```
bool R_MTU3_ControlUnit(
    uint8_t data1,           // Unit selection
    R_MTU3_ControlUnit_structure ptr // A pointer to the structure
);
```

R\_MTU3\_ControlUnit\_structure members:

```
uint16_t data2 // Stop / Start control selection
uint16_t data3 // Channel pair control selection and options
uint32_t data4 // Channel-pair-specific control selection
uint32_t data5 // Phase output control selection
uint32_t data6 // Phase output buffer control selection
uint32_t data7 // Interrupt skipping control
uint16_t data8 // Register value
uint16_t data9 // Register value
uint16_t data10 // Register value
```

### Description (1/6)

Modify the timer unit's registers.

#### [data1]

The unit number n (where n = 0).

#### [data2]

Simultaneous stop / start control. All selections are optional.  
If multiple selections are required, use “|” to separate each selection.  
Specify PDL\_NO\_DATA if no change is required.

#### • Counter stop control

PDL_MTU3_STOP_CH_0	Stop the count operation for the selected channels.
PDL_MTU3_STOP_CH_1	
PDL_MTU3_STOP_CH_2	
PDL_MTU3_STOP_CH_3	
PDL_MTU3_STOP_CH_4	
PDL_MTU3_STOP_CH_6	
PDL_MTU3_STOP_CH_7	

#### • Counter start control

PDL_MTU3_START_CH_0	Start the count operation for the selected channels. The start will be simultaneous.
PDL_MTU3_START_CH_1	
PDL_MTU3_START_CH_2	
PDL_MTU3_START_CH_3	
PDL_MTU3_START_CH_4	
PDL_MTU3_START_CH_6	
PDL_MTU3_START_CH_7	

**Description (2/6)****[data3]**

Channel pair configuration control.

Select the pair of channels that will be configured by the controls specified below and in parameters data4 to data7.

- Channel pair selection

PDL_MTU3_CONTROL_CH_34	Configure the operation of channels 3 and 4.
PDL_MTU3_CONTROL_CH_67	Configure the operation of channels 6 and 7.

Select the controls. All selections are optional.

If identical controls are required for both pairs, use "I" to separate each selection.

Specify PDL\_NO\_DATA if no change is required.

- Register protection

PDL_MTU3_ACCESS_DISABLE	Control the access to some control registers and counters.
PDL_MTU3_ACCESS_ENABLE	

- Dead time generation control

PDL_MTU3_DEAD_TIME_DISABLE or PDL_MTU3_DEAD_TIME_ENABLE	Disable or enable dead time generation.
--	---

- Waveform retention control

PDL_MTU3_WAVEFORM_RETAIN_DISABLE or PDL_MTU3_WAVEFORM_RETAIN_ENABLE	Disable or enable waveform output retention.
--	--

- Synchronous clearing control

Applies only to channel pair 6 and 7.

PDL_MTU3_SYNC_CLEAR_DISABLE or PDL_MTU3_SYNC_CLEAR_ENABLE	Disable or enable synchronous clearing.
--	---

- Compare match clearing control

PDL_MTU3_CNT_CLEAR_CM_A_DISABLE or PDL_MTU3_CNT_CLEAR_CM_A_ENABLE	Disable or enable counter clearing on TGRA compare match.
--	---

- Reset-synchronised or complementary PWM control

PDL_MTU3_PWM_RS_COMP_ENABLE	Enable reset-synchronised or complementary PWM mode.
-----------------------------	--

- Registers to be modified.

PDL_MTU3_REGISTER_DEAD_TIME	Update the dead time data register (TDDR) using the value supplied in parameter data8.
PDL_MTU3_REGISTER_CYCLE_DATA	Update the cycle data register (TCDR) using the value supplied in parameter data9.
PDL_MTU3_REGISTER_CYCLE_BUFFER	Update the cycle buffer register (TCBR) using the value supplied in parameter data10.

**Description (3/6)****[data4]**

The control settings which are specific to one pair of channels. All settings are optional.  
If multiple selections are required, use “|” to separate each selection.  
Specify PDL\_NO\_DATA if no change is required.

- Brushless DC motor control (applies only to channel pair 3 and 4)

PDL_MTU3_BDCM_ENABLE or PDL_MTU3_BDCM_DISABLE	Enable or disable brushless DC motor control
PDL_MTU3_BDCM_P_PHASE_ENABLE or PDL_MTU3_BDCM_P_PHASE_DISABLE	Enable or disable PWM outputs on the positive-phase output pins.
PDL_MTU3_BDCM_N_PHASE_ENABLE or PDL_MTU3_BDCM_N_PHASE_DISABLE	Enable or disable PWM outputs on the negative-phase output pins.
PDL_MTU3_BDCM_OPS_FB or	Use input capture signals for output switch control, or
PDL_MTU3_BDCM_OPS_000 or PDL_MTU3_BDCM_OPS_001 or PDL_MTU3_BDCM_OPS_010 or PDL_MTU3_BDCM_OPS_011 or PDL_MTU3_BDCM_OPS_100 or PDL_MTU3_BDCM_OPS_101 or PDL_MTU3_BDCM_OPS_110 or PDL_MTU3_BDCM_OPS_111	Set the outputs according to table 15.47 in the hardware manual.

- Synchronous clearing control (applies only to channel pair 6 and 7)

PDL_MTU3_SYNC_CLEAR_TGRB2_DISABLE or PDL_MTU3_SYNC_CLEAR_TGRB2_ENABLE	Disable or enable clearing on channel 2 TGRB input capture or compare match.
PDL_MTU3_SYNC_CLEAR_TGRA2_DISABLE or PDL_MTU3_SYNC_CLEAR_TGRA2_ENABLE	Disable or enable clearing on channel 2 TGRA input capture or compare match.
PDL_MTU3_SYNC_CLEAR_TGRB1_DISABLE or PDL_MTU3_SYNC_CLEAR_TGRB1_ENABLE	Disable or enable clearing on channel 1 TGRB input capture or compare match.
PDL_MTU3_SYNC_CLEAR_TGRA1_DISABLE or PDL_MTU3_SYNC_CLEAR_TGRA1_ENABLE	Disable or enable clearing on channel 1 TGRA input capture or compare match.
PDL_MTU3_SYNC_CLEAR_TGRD0_DISABLE or PDL_MTU3_SYNC_CLEAR_TGRD0_ENABLE	Disable or enable clearing on channel 0 TGRD input capture or compare match.
PDL_MTU3_SYNC_CLEAR_TGRC0_DISABLE or PDL_MTU3_SYNC_CLEAR_TGRC0_ENABLE	Disable or enable clearing on channel 0 TGRC input capture or compare match.
PDL_MTU3_SYNC_CLEAR_TGRB0_DISABLE or PDL_MTU3_SYNC_CLEAR_TGRB0_ENABLE	Disable or enable clearing on channel 0 TGRB input capture or compare match.
PDL_MTU3_SYNC_CLEAR_TGRA0_DISABLE or PDL_MTU3_SYNC_CLEAR_TGRA0_ENABLE	Disable or enable clearing on channel 0 TGRA input capture or compare match.

**Description (4/6)****[data5]**

The phase output control settings to be modified. All settings are optional.  
If multiple selections are required, use “|” to separate each selection.  
Specify PDL\_NO\_DATA if no change is required.

- Output enable control. To apply output control, make sure the operation of the corresponding channel is stopped.  
Specify PDL\_NO\_DATA if no change is required.  
Select one option for each output.

PDL_MTU3_OUT_P_PHASE_1_ENABLE or PDL_MTU3_OUT_P_PHASE_1_DISABLE	For channels 3 and 4: control MTIOC3B. For channels 6 and 7: control MTIOC6B.
PDL_MTU3_OUT_N_PHASE_1_ENABLE or PDL_MTU3_OUT_N_PHASE_1_DISABLE	For channels 3 and 4: control MTIOC3D. For channels 6 and 7: control MTIOC6D.
PDL_MTU3_OUT_P_PHASE_2_ENABLE or PDL_MTU3_OUT_P_PHASE_2_DISABLE	For channels 3 and 4: control MTIOC4A. For channels 6 and 7: control MTIOC7A.
PDL_MTU3_OUT_N_PHASE_2_ENABLE or PDL_MTU3_OUT_N_PHASE_2_DISABLE	For channels 3 and 4: control MTIOC4C. For channels 6 and 7: control MTIOC7C.
PDL_MTU3_OUT_P_PHASE_3_ENABLE or PDL_MTU3_OUT_P_PHASE_3_DISABLE	For channels 3 and 4: control MTIOC4B. For channels 6 and 7: control MTIOC7B.
PDL_MTU3_OUT_N_PHASE_3_ENABLE or PDL_MTU3_OUT_N_PHASE_3_DISABLE	For channels 3 and 4: control MTIOC4D. For channels 6 and 7: control MTIOC7D.

Or all six phase outputs can be controlled together by selecting one of each:

PDL_MTU3_OUT_P_PHASE_ALL_ENABLE or PDL_MTU3_OUT_P_PHASE_ALL_DISABLE	All P phase outputs.
PDL_MTU3_OUT_N_PHASE_ALL_ENABLE or PDL_MTU3_OUT_N_PHASE_ALL_DISABLE	All N phase outputs.

- Output inversion control.  
Each phase output can be configured for  
a) initial high level, active low level or  
b) initial low level, active high level.

All six phase outputs can be controlled together by selecting one of each:

PDL_MTU3_OUT_P_PHASE_ALL_HIGH_LOW or PDL_MTU3_OUT_P_PHASE_ALL_LOW_HIGH	Positive-phase outputs.
PDL_MTU3_OUT_N_PHASE_ALL_HIGH_LOW or PDL_MTU3_OUT_N_PHASE_ALL_LOW_HIGH	Negative-phase outputs.

Or independently by selecting one option for each required output.

PDL_MTU3_OUT_P_PHASE_1_HIGH_LOW or PDL_MTU3_OUT_P_PHASE_1_LOW_HIGH	The same outputs as listed for Output enable control.
PDL_MTU3_OUT_N_PHASE_1_HIGH_LOW or PDL_MTU3_OUT_N_PHASE_1_LOW_HIGH	
PDL_MTU3_OUT_P_PHASE_2_HIGH_LOW or PDL_MTU3_OUT_P_PHASE_2_LOW_HIGH	
PDL_MTU3_OUT_N_PHASE_2_HIGH_LOW or PDL_MTU3_OUT_N_PHASE_2_LOW_HIGH	
PDL_MTU3_OUT_P_PHASE_3_HIGH_LOW or PDL_MTU3_OUT_P_PHASE_3_LOW_HIGH	
PDL_MTU3_OUT_N_PHASE_3_HIGH_LOW or PDL_MTU3_OUT_N_PHASE_3_LOW_HIGH	

- Write access control

PDL_MTU3_OUT_LOCK_ENABLE	Prevent further changes to the phase output control.
--------------------------	--

- PWM synchronous output control

PDL_MTU3_OUT_TOGGLE_ENABLE or PDL_MTU3_OUT_TOGGLE_DISABLE	Enable or disable toggle output synchronised with the PWM cycle.
--	--

**Description (5/6)****[data6]**

The phase output buffer control settings to be modified. All settings are optional.  
 If multiple selections are required, use “|” to separate each selection.  
 Specify PDL\_NO\_DATA if no change is required.

- Output level buffer control

Set the output control to be transferred to the output:

PDL_MTU3_OUT_BUFFER_P_PHASE_1_LOW or PDL_MTU3_OUT_BUFFER_P_PHASE_1_HIGH	Buffer control for MTIOC3B or MTIOC6B.
PDL_MTU3_OUT_BUFFER_N_PHASE_1_LOW or PDL_MTU3_OUT_BUFFER_N_PHASE_1_HIGH	Buffer control for MTIOC3D or MTIOC6D.
PDL_MTU3_OUT_BUFFER_P_PHASE_2_LOW or PDL_MTU3_OUT_BUFFER_P_PHASE_2_HIGH	Buffer control for MTIOC4A or MTIOC7A.
PDL_MTU3_OUT_BUFFER_N_PHASE_2_LOW or PDL_MTU3_OUT_BUFFER_N_PHASE_2_HIGH	Buffer control for MTIOC4C or MTIOC7C.
PDL_MTU3_OUT_BUFFER_P_PHASE_3_LOW or PDL_MTU3_OUT_BUFFER_P_PHASE_3_HIGH	Buffer control for MTIOC4B or MTIOC7B.
PDL_MTU3_OUT_BUFFER_N_PHASE_3_LOW or PDL_MTU3_OUT_BUFFER_N_PHASE_3_HIGH	Buffer control for MTIOC4D or MTIOC7D.

- Set the transfer timing

In complementary PWM mode:

PDL_MTU3_OUT_BUFFER_TRANSFER_DISABLE or PDL_MTU3_OUT_BUFFER_TRANSFER_CREST or PDL_MTU3_OUT_BUFFER_TRANSFER_TROUGH or PDL_MTU3_OUT_BUFFER_TRANSFER_BOTH	Disable or enable on detection of crest, trough or both
---	--

In Reset-synchronised PWM mode:

PDL_MTU3_OUT_BUFFER_TRANSFER_DISABLE or PDL_MTU3_OUT_BUFFER_TRANSFER_CLEAR	Disable or enable on counter clear.
---	-------------------------------------

- Buffer transfer to temporary transfer control

PDL_MTU3_BUFFER_TRANSFER_DISABLE or PDL_MTU3_BUFFER_TRANSFER_ENABLE or PDL_MTU3_BUFFER_TRANSFER_LINK	Disable transfers, enable without linking to interrupt skipping or enable and link to interrupt skipping.
--	--

**Description (6/6)****[data7]**

Interrupt skipping control settings. All settings are optional, but only one skipping function type may be selected.

If multiple selections are required, use “|” to separate each selection.

Specify PDL\_NO\_DATA if no change is required.

- Interrupt skipping control (type 1)

PDL_MTU3_INT_SKIP_TROUGH_DISABLE or PDL_MTU3_INT_SKIP_TROUGH_1 or PDL_MTU3_INT_SKIP_TROUGH_2 or PDL_MTU3_INT_SKIP_TROUGH_3 or PDL_MTU3_INT_SKIP_TROUGH_4 or PDL_MTU3_INT_SKIP_TROUGH_5 or PDL_MTU3_INT_SKIP_TROUGH_6 or PDL_MTU3_INT_SKIP_TROUGH_7	Disable TCNT underflow (TCIV4 or TCIV7) interrupt skipping, or set the skip count between 1 and 7.
PDL_MTU3_INT_SKIP_CREST_DISABLE or PDL_MTU3_INT_SKIP_CREST_1 or PDL_MTU3_INT_SKIP_CREST_2 or PDL_MTU3_INT_SKIP_CREST_3 or PDL_MTU3_INT_SKIP_CREST_4 or PDL_MTU3_INT_SKIP_CREST_5 or PDL_MTU3_INT_SKIP_CREST_6 or PDL_MTU3_INT_SKIP_CREST_7	Disable TGRA compare match (TGIA3 or TGIA6) interrupt skipping, or set the skip count between 1 and 7.

- Interrupt skipping control (type 2)

PDL_MTU3_INT_SKIP_ADC_DISABLE or PDL_MTU3_INT_SKIP_ADC_1 or PDL_MTU3_INT_SKIP_ADC_2 or PDL_MTU3_INT_SKIP_ADC_3 or PDL_MTU3_INT_SKIP_ADC_4 or PDL_MTU3_INT_SKIP_ADC_5 or PDL_MTU3_INT_SKIP_ADC_6 or PDL_MTU3_INT_SKIP_ADC_7	Disable ADC trigger TRGnAN and TRGnBN skipping, or set the skip count between 1 and 7. If channels 3 and 4 have been selected, n = 4. If channels 6 and 7 have been selected, n = 7.
---	--

**[data8]**

The dead time data register value. This will be ignored if the register is not selected.

**[data9]**

The cycle data register value. This will be ignored if the register is not selected.

**[data10]**

The cycle buffer register value. This will be ignored if the register is not selected.

**Return value**

True if all parameters are valid and exclusive; otherwise false.

**Category**

Multi-function Timer Pulse Unit

**Reference**

R\_MTU3\_ControlChannel, R\_MTU3\_Create

**Remarks**

- Either this function or R\_MTU3\_ControlChannel must be used to start the timers.
- The Stop operation is executed at the start of this function.  
The Start operation is executed at the end.  
Therefore, both options can be selected together with other changes in one function call.
- The register access enable operation is executed at the start of this function.  
The register access disable operation is executed at the end.  
Therefore, both options can be selected together with other changes in one function call.
- The enabling of reset-synchronised or complementary PWM mode is made after all other configuration settings are made and before the timers are started. R\_MTU3\_Create must be used first to configure the timer channel (3 or 6).
- A companion function, R\_MTU3\_ControlUnit\_load\_defaults, can be used to load the default values into the structure.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_mtu3.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Allocate a copy of the structure */
    R_MTU3_ControlUnit_structure unit_parameters;

    /* Load the defaults */
    R_MTU3_ControlUnit_load_defaults(&unit_parameters);

    /* Set the control options for unit 0 */
    unit_parameters.data2 = PDL_MTU3_START_CH_0 | PDL_MTU3_START_CH_1;
    unit_parameters.data3 = PDL_MTU3_DEAD_TIME_ENABLE |
PDL_MTU3_REGISTER_DEAD_TIME | PDL_MTU3_REGISTER_CYCLE_DATA;
    unit_parameters.data5 = PDL_MTU3_OUT_P_PHASE_ALL_HIGH_LOW;
    unit_parameters.data8 = 0xFFDD;
    unit_parameters.data9 = 0x0100;

    /* Modify the operation of unit 0 */
    R_MTU3_ControlUnit(
        0,
        &unit_parameters
    );
}
```



## 6) R\_MTU3\_ReadChannel

### Synopsis

Read from MTU channel registers.

### Prototype

```
bool R_MTU3_ReadChannel(
    uint8_t data1,      // Channel selection
    uint8_t * data2,     // A pointer to the data storage location
    uint16_t * data3,    // A pointer to the data storage location
    uint16_t * data4,    // A pointer to the data storage location
    uint16_t * data5,    // A pointer to the data storage location
    uint16_t * data6,    // A pointer to the data storage location
    uint16_t * data7,    // A pointer to the data storage location
    uint16_t * data8,    // A pointer to the data storage location
    uint16_t * data9     // A pointer to the data storage location
);
```

### Description (1/2)

Read any of the timer's counter, compare or status flag registers.

#### [data1]

The channel number n (where n = 0 to 7).

#### [data2]

The status flags shall be stored in the format below.

The input capture / compare match flags will be set to 1 if the condition has been detected.

Specify PDL\_NO\_PTR if the flags are not to be read.

For n = 0

	b7	b6	b5	b4	b3	b2	b1	b0
0	Detection							
	Overflow		Input capture / compare match					
	V		F	E	D	C	B	A

For n = 1 and 2

	b7	b6	b5	b4 - b2	b1	b0
Count direction	Detection					
	Overflow		Underflow	0	Input capture / compare match	
	V		U		B	A
0: down 1: up						

For n = 3 and 6

b7		b6		b5 - b4		b3	b2	b1	b0
Count direction	Detection								
	Overflow		0	Input capture / compare match					
	V			D	C	B	A		
0: down 1: up									

For n = 4 and 7

b7		b6		b5 - b4		b3	b2	b1	b0
Count direction	Detection								
	Overflow or underflow			0	Input capture / compare match				
	V				D	C	B	A	
0: down 1: up									

For n = 5

	b7 - b3	b2	b1	b0
0	Detection			
	Input capture / compare match			
	U	V	W	

#### [data3]

For n ≠ 5: A pointer to where the TNCT register value shall be stored.

For n = 5: A pointer to where the TNCTU register value shall be stored.

Specify PDL\_NO\_PTR if it is not required.

<b>Description (2/2)</b>	<p><b>[data4]</b>  For n ≠ 5: A pointer to where the TGRA register value shall be stored.  For n = 5: A pointer to where the TNCTV register value shall be stored.  Specify PDL_NO_PTR if it is not required.</p> <p><b>[data5]</b>  For n ≠ 5: A pointer to where the TGRB register value shall be stored.  For n = 5: A pointer to where the TNCTW register value shall be stored.  Specify PDL_NO_PTR if it is not required.</p> <p><b>[data6]</b>  For n = 0, 3, 4, 6 and 7: A pointer to where the TGRC register value shall be stored.  For n = 5: A pointer to where the TGRU register value shall be stored.  Specify PDL_NO_PTR if it is not required.</p> <p><b>[data7]</b>  For n = 0, 3, 4, 6 and 7: A pointer to where the TGRD register value shall be stored.  For n = 5: A pointer to where the TGRV register value shall be stored.  Specify PDL_NO_PTR if it is not required.</p> <p><b>[data8]</b>  For n = 0, 3, 4, 6 and 7: A pointer to where the TGRE register value shall be stored.  For n = 5: A pointer to where the TGRW register value shall be stored.  Specify PDL_NO_PTR if it is not required.</p> <p><b>[data9]</b>  For n = 0, 4 and 7: A pointer to where the TGRF register value shall be stored.  Specify PDL_NO_PTR if it is not required.</p>
<b>Return value</b>	True if all parameters are valid and exclusive; otherwise false.
<b>Category</b>	Multi-function Timer Pulse Unit
<b>Reference</b>	None.
<b>Remarks</b>	<ul style="list-style-type: none"> <li>If the flags are read, any detection flag that has been set to 1 shall be automatically cleared to 0 by this function.</li> </ul>
<b>Program example</b>	<pre> /* RPDL definitions */ #include "r_pdl_mtu3.h"  /* RPDL device-specific definitions */ #include "r_pdl_definitions.h"  uint8_t Flags; uint16_t General_A; uint16_t General_D;  void func(void) {     /* Read the status flags and registers of channel 3 */     R_MTU3_ReadChannel(         3,         &amp;Flags,         PDL_NO_PTR,         &amp;General_A,         PDL_NO_PTR,         PDL_NO_PTR,         &amp;General_D,         PDL_NO_PTR,         PDL_NO_PTR     ); } </pre>

## 7) R\_MTU3\_ReadUnit

### Synopsis

Read from MTU unit registers.

### Prototype

```
bool R_MTU3_ReadUnit(
    uint8_t data1,    // Unit selection
    uint8_t data2,    // Register selection
    uint16_t * data3, // A pointer to the data storage location
    uint8_t * data4   // A pointer to the data storage location
);
```

### Description

Read any of the timer units's counter registers

#### [data1]

The unit number n (where n = 0).

#### [data2]

- Channel pair selection

PDL_MTU3_CH_34 or PDL_MTU3_CH_67	Read the A registers (used with channels 3 and 4) or the B registers (used with channels 6 and 7).
-------------------------------------	---

#### [data3]

A pointer to where the Timer Subcounter (TCNTS) register value shall be stored. Specify PDL\_NO\_PTR if it is not required.

#### [data4]

Where the Timer Interrupt Skipping Counters (TITCNT1 or TITCNT2) register value shall be stored. The choice of register depends on the type on interrupt skipping control selected by function R\_MTU3\_ControlUnit. Specify PDL\_NO\_PTR if it is not required.

### Return value

True if all parameters are valid and exclusive; otherwise false.

### Category

Multi-function Timer Pulse Unit

### Reference

R\_MTU3\_ControlUnit

### Remarks

- None.

### Program example

```
/* RPDL definitions */
#include "r_pdl_mtu3.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

uint16_t Sub_count;
uint8_t Skip_count;

void func(void)
{
    /* Read the counter registers for channels 3 and 4 */
    R_MTU3_ReadUnit(
        0,
        PDL_MTU3_CH_34,
        &Sub_count,
        &Skip_count
    );
}
```

## 4.2.11. Port Output Enable

## 1) R\_POE\_Set

**Synopsis**

Configure the Port Output Enable module.

**Prototype**

```
bool R_POE_Set(
    uint32_t data1, // Input pin configuration
    uint32_t data2, // High impedance control
    uint16_t data3, // Output pin selection
    uint16_t data4  // Output short detection and response
);
```

**Description (1/3)**

Initialise the POE pins.

**[data1]**

Configure the input pin detection for pins POE0, POE4, POE8, POE10 and POE11.  
 If multiple selections are required, use “|” to separate each selection.  
 All selections are optional. Specify PDL\_NO\_DATA if none are required.

PDL_POE_0_MODE_EDGE or PDL_POE_0_MODE_LOW_8 or PDL_POE_0_MODE_LOW_16 or PDL_POE_0_MODE_LOW_128	For any pin POEn, select falling edge or low level for 16 samples at PCLK ÷ 8, 16 or 128.
PDL_POE_4_MODE_EDGE or PDL_POE_4_MODE_LOW_8 or PDL_POE_4_MODE_LOW_16 or PDL_POE_4_MODE_LOW_128	
PDL_POE_8_MODE_EDGE or PDL_POE_8_MODE_LOW_8 or PDL_POE_8_MODE_LOW_16 or PDL_POE_8_MODE_LOW_128	
PDL_POE_10_MODE_EDGE or PDL_POE_10_MODE_LOW_8 or PDL_POE_10_MODE_LOW_16 or PDL_POE_10_MODE_LOW_128	
PDL_POE_11_MODE_EDGE or PDL_POE_11_MODE_LOW_8 or PDL_POE_11_MODE_LOW_16 or PDL_POE_11_MODE_LOW_12	

• Pin selection.

PDL_POE_10_A or PDL_POE_10_B	Select the -A or -B pin for POE10. Not required if input POE10 is disabled.
---------------------------------	--

**Description (2/3)****[data2]**

High impedance control selections.

If multiple selections are required, use “|” to separate each selection.

All selections are optional. Specify PDL\_NO\_DATA if none are required.

- High impedance request detection

PDL_POE_HI_Z_REQ_8_ENABLE	Enable high-impedance requests on pin POE8.
PDL_POE_HI_Z_REQ_10_ENABLE	Enable high-impedance requests on pin POE10.
PDL_POE_HI_Z_REQ_11_ENABLE	Enable high-impedance requests on pin POE11.

- Select any event flags to be added to the high-impedance control for the specified outputs.

PDL_POE_HI_Z_MT34_ADD_CFLAG	Comparator detection		MTU channel 3 and 4 and GPT channel 0 to 2 outputs.
PDL_POE_HI_Z_MT34_ADD_POE4	A valid edge on the pin:	POE4	
PDL_POE_HI_Z_MT34_ADD_POE8		POE8	
PDL_POE_HI_Z_MT34_ADD_POE10		POE10	
PDL_POE_HI_Z_MT34_ADD_POE11		POE11	

PDL_POE_HI_Z_MT67_ADD_CFLAG	Comparator detection		MTU channel 6 and 7 outputs.
PDL_POE_HI_Z_MT67_ADD_POE0	A valid edge on the pin:	POE0	
PDL_POE_HI_Z_MT67_ADD_POE8		POE8	
PDL_POE_HI_Z_MT67_ADD_POE10		POE10	
PDL_POE_HI_Z_MT67_ADD_POE11		POE11	

PDL_POE_HI_Z_MT0_ADD_CFLAG	Comparator detection		MTU channel 0 outputs.
PDL_POE_HI_Z_MT0_ADD_POE0	A valid edge on the pin:	POE0	
PDL_POE_HI_Z_MT0_ADD_POE4		POE4	
PDL_POE_HI_Z_MT0_ADD_POE10		POE10	
PDL_POE_HI_Z_MT0_ADD_POE11		POE11	

PDL_POE_HI_Z_GPT01_ADD_CFLAG	Comparator detection		GPT channel 0 and 1 outputs.
PDL_POE_HI_Z_GPT01_ADD_POE0	A valid edge on the pin:	POE0	
PDL_POE_HI_Z_GPT01_ADD_POE4		POE4	
PDL_POE_HI_Z_GPT01_ADD_POE8		POE8	
PDL_POE_HI_Z_GPT01_ADD_POE11		POE11	

PDL_POE_HI_Z_GPT23_ADD_CFLAG	Comparator detection		GPT channel 2 and 3 outputs.
PDL_POE_HI_Z_GPT23_ADD_POE0	A valid edge on the pin:	POE0	
PDL_POE_HI_Z_GPT23_ADD_POE4		POE4	
PDL_POE_HI_Z_GPT23_ADD_POE8		POE8	
PDL_POE_HI_Z_GPT23_ADD_POE10		POE10	

**Description (3/3)****[data3]**

Select the output pins to be controlled.

If multiple selections are required, use “|” to separate each selection.

All selections are optional. Specify PDL\_NO\_DATA if none are required.

PDL_POE_HI_Z_ENABLE_MTI0C0A	MTU channel 0.
PDL_POE_HI_Z_ENABLE_MTI0C0B	Input pin or event high impedance request, software control or the oscillation stop detection flag.
PDL_POE_HI_Z_ENABLE_MTI0C0C	
PDL_POE_HI_Z_ENABLE_MTI0C0D	

PDL_POE_HI_Z_ENABLE_MTI0C7BD	MTU channel 6 and 7.
PDL_POE_HI_Z_ENABLE_MTI0C7AC	Input pin or event high impedance request, output short detection, software control or the oscillation stop detection flag.
PDL_POE_HI_Z_ENABLE_MTI0C6BD	

PDL_POE_HI_Z_ENABLE_MTI0C4BD	MTU channel 3 and 4.
PDL_POE_HI_Z_ENABLE_MTI0C4AC	Input pin or event high impedance request, output short detection, software control or the oscillation stop detection flag.
PDL_POE_HI_Z_ENABLE_MTI0C3BD	

PDL_POE_HI_Z_ENABLE_GTIO0C0	GPT channel pins A and B.
PDL_POE_HI_Z_ENABLE_GTIO0C1	Input pin or event high impedance request, software control or the oscillation stop detection flag.
PDL_POE_HI_Z_ENABLE_GTIO0C2	
PDL_POE_HI_Z_ENABLE_GTIO0C3	

**[data4]**

Output short detection and response. All selections are optional.

If multiple selections are required, use “|” to separate each selection.

The default settings are shown in **bold**. Specify PDL\_NO\_DATA to use the defaults.

- Output short detection level

<b>PDL_POE_SHORT_USE_MTU</b> or PDL_POE_SHORT_SPECIFY	Use the active levels specified in the MTU3 functions, or enable the settings below.
<b>PDL_POE_SHORT_P71_LOW</b> or PDL_POE_SHORT_P71_HIGH	Select the port pin active level for detection of short circuits. Ignored if the MTU3 settings are used.
<b>PDL_POE_SHORT_P74_LOW</b> or PDL_POE_SHORT_P74_HIGH	
<b>PDL_POE_SHORT_P72_LOW</b> or PDL_POE_SHORT_P72_HIGH	
<b>PDL_POE_SHORT_P75_LOW</b> or PDL_POE_SHORT_P75_HIGH	
<b>PDL_POE_SHORT_P73_LOW</b> or PDL_POE_SHORT_P73_HIGH	
<b>PDL_POE_SHORT_P76_LOW</b> or PDL_POE_SHORT_P76_HIGH	

- Output short response

PDL_POE_SHORT_P7X_HI_Z	If a short is detected, place the all the selected MTU channel 3 and 4 (or GPT channel 0 to 2) pins in the high impedance state.
PDL_POE_SHORT_MTU_67_HI_Z	If a short is detected, place the all the selected MTU channel 6 and 7 pins in the high impedance state.

**Return value**

True if all parameters are valid and exclusive; otherwise false.

**Category**

Port Output Enable

**Reference**

R\_POE\_Control, R\_MTU3\_Set, R\_INTC\_GetExtInterruptStatus

**Remarks**

- Pins POE4, POE10-B and POE11 are not available on the 64-pin package.
- Do not select output pins that are not used.
- The oscillation stop detection flag may be read using R\_INTC\_GetExtInterruptStatus.

**Program example**

```

/* RPDL definitions */
#include "r_pdl_poe.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Configure POE pins */
    R_POE_Set(
        PDL_POE_0_MODE_EDGE      | PDL_POE_4_MODE_LOW_8 | \
        PDL_POE_8_MODE_LOW_16    | PDL_POE_10_MODE_LOW_128 | \
        PDL_POE_10_B,
        PDL_POE_HI_Z_REQ_8_ENABLE | \
        PDL_POE_HI_Z_MT34_ADD_CFLAG | PDL_POE_HI_Z_MT67_ADD_POE0 | \
        PDL_POE_HI_Z_MT0_ADD_POE4 | PDL_POE_HI_Z_GPT01_ADD_POE11,
        PDL_POE_HI_Z_ENABLE_MTIOC0A | PDL_POE_HI_Z_ENABLE_MTIOC7AC | \
        PDL_POE_HI_Z_ENABLE_MTIOC3BD | PDL_POE_HI_Z_ENABLE_GTIOC3,
        PDL_POE_SHORT_SPECIFY | PDL_POE_SHORT_P71_HIGH | \
        PDL_POE_SHORT_P7X_HI_Z | PDL_POE_SHORT_MTU_67_HI_Z
    );
}

```

## 2) R\_POE\_Create

### Synopsis

Configure the Port Output Enable event handling.

### Prototype

```
bool R_POE_Create(
    uint16_t data1,    // Input configuration selection
    void * func1,      // Callback function
    void * func2,      // Callback function
    void * func3,      // Callback function
    void * func4,      // Callback function
    uint8_t data2      // Interrupt priority level
);
```

### Description

Enable interrupts and register the callback functions.

#### [data1]

Interrupt selection.

If multiple selections are required, use “|” to separate each selection.

The default settings are shown in **bold**. Specify PDL\_NO\_DATA to use the defaults.

- High impedance request response

<b>PDL_POE_IRQ_HI_Z_0_DISABLE</b> or PDL_POE_IRQ_HI_Z_0_ENABLE	Disable or enable an interrupt on detection of a high impedance request on pin POE0.
<b>PDL_POE_IRQ_HI_Z_4_DISABLE</b> or PDL_POE_IRQ_HI_Z_4_ENABLE	Disable or enable an interrupt on detection of a high impedance request on pin POE4.
<b>PDL_POE_IRQ_HI_Z_10_DISABLE</b> or PDL_POE_IRQ_HI_Z_10_ENABLE	Disable or enable an interrupt on detection of a high impedance request on pin POE10.
<b>PDL_POE_IRQ_HI_Z_11_DISABLE</b> or PDL_POE_IRQ_HI_Z_11_ENABLE	Disable or enable an interrupt on detection of a high impedance request on pin POE11.

- Output short detection response

<b>PDL_POE_IRQ_SHORT_34_DISABLE</b> or PDL_POE_IRQ_SHORT_34_ENABLE	Disable or enable an interrupt on detection of a short on any MTU channel 3 or 4 two-phase output pair.
<b>PDL_POE_IRQ_SHORT_67_DISABLE</b> or PDL_POE_IRQ_SHORT_67_ENABLE	Disable or enable an interrupt on detection of a short on any MTU channel 6 or 7 two-phase output pair.

#### [func1]

The function to be called when an enabled request on pin POE0 or an output short on MTU channels 3 or 4 occurs.

Specify PDL\_NO\_FUNC if not required.

#### [func2]

The function to be called when an enabled request on pin POE4 or an output short on MTU channels 6 or 7 occurs.

Specify PDL\_NO\_FUNC if not required.

#### [func3]

The function to be called when an enabled request on pin POE8 occurs.

Specify PDL\_NO\_FUNC if not required.

#### [func4]

The function to be called when an enabled request on pin POE10 or POE11 occurs.

Specify PDL\_NO\_FUNC if not required.

#### [data2]

The interrupt priority level. Select between 1 (lowest priority) and 15 (highest priority).

This parameter will be ignored if PDL\_NO\_FUNC is specified for all parameters func1 to func4.

### Return value

True if all parameters are valid and exclusive; otherwise false.

### Category

Port Output Enable



**Reference**

R\_POE\_GetStatus

**Remarks**

- A callback function is executed by the interrupt processing function. This means that no other interrupt can be processed until the callback function has completed.
- Use R\_POE\_GetStatus to determine the interrupt cause.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_poe.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void POE0_handler(void){}

void func(void)
{
    /* Assign the callback function for pin POE0 */
    R_POE_Create(
        PDL_POE_IRQ_HI_Z_0_ENABLE,
        POE0_handler,
        PDL_NO_FUNC,
        PDL_NO_FUNC,
        PDL_NO_FUNC,
        15
    );
}
```

### 3) R\_POE\_Control

#### Synopsis

Control the Port Output Enable module.

#### Prototype

```
bool R_POE_Control (
    uint16_t data1, // Control options
    uint8_t data2,  // Control options
    uint16_t data3  // Control options
);
```

#### Description

Change the state of output pins, status flags and interrupt control.

##### [data1]

Manual high impedance control.

If multiple selections are required, use “|” to separate each selection.

All settings are optional. Specify PDL\_NO\_DATA if no control is required.

- MTU / GPT channel output high impedance control

PDL_POE_MTU34_HI_Z_ON or PDL_POE_MTU34_HI_Z_OFF	Control the high impedance state of the MTU channel 3 and 4 outputs.
PDL_POE_MTU67_HI_Z_ON or PDL_POE_MTU67_HI_Z_OFF	Control the high impedance state of the MTU channel 6 and 7 outputs.
PDL_POE_MTU0_HI_Z_ON or PDL_POE_MTU0_HI_Z_OFF	Control the high impedance state of the MTU channel 0 outputs.
PDL_POE_GPT01_HI_Z_ON or PDL_POE_GPT01_HI_Z_OFF	Control the high impedance state of the GPT channel 0 and 1 outputs.
PDL_POE_GPT23_HI_Z_ON or PDL_POE_GPT23_HI_Z_OFF	Control the high impedance state of the GPT channel 2 and 3 outputs.

##### [data2]

Event flag control.

If multiple selections are required, use “|” to separate each selection.

All settings are optional. Specify PDL\_NO\_DATA if no control is required.

PDL_POE_FLAG_POE0_CLEAR	Select the flags to be cleared.
PDL_POE_FLAG_POE4_CLEAR	
PDL_POE_FLAG_POE8_CLEAR	
PDL_POE_FLAG_POE10_CLEAR	
PDL_POE_FLAG_POE11_CLEAR	
PDL_POE_FLAG_SHORT_34_CLEAR	
PDL_POE_FLAG_SHORT_67_CLEAR	

##### [data3]

Interrupt control. If multiple selections are required, use “|” to separate each selection.

All settings are optional. Specify PDL\_NO\_DATA if no control is required.

- High impedance request response

PDL_POE_IRQ_HI_Z_0_DISABLE	Control interrupts on detection of a high impedance request on pin POE0.
PDL_POE_IRQ_HI_Z_0_ENABLE	
PDL_POE_IRQ_HI_Z_4_DISABLE	Control interrupts on detection of a high impedance request on pin POE4.
PDL_POE_IRQ_HI_Z_4_ENABLE	
PDL_POE_IRQ_HI_Z_8_DISABLE	Control interrupts on detection of a high impedance request on pin POE8.
PDL_POE_IRQ_HI_Z_8_ENABLE	
PDL_POE_IRQ_HI_Z_10_DISABLE	Control interrupts on detection of a high impedance request on pin POE10.
PDL_POE_IRQ_HI_Z_10_ENABLE	
PDL_POE_IRQ_HI_Z_11_DISABLE	Control interrupts on detection of a high impedance request on pin POE11.
PDL_POE_IRQ_HI_Z_11_ENABLE	

- Output short detection response

PDL_POE_IRQ_SHORT_34_DISABLE	Control interrupts on detection of a short on any MTU channel 3 or 4 two-phase output pair.
PDL_POE_IRQ_SHORT_34_ENABLE	
PDL_POE_IRQ_SHORT_67_DISABLE	Control interrupts on detection of a short on any MTU channel 6 or 7 two-phase output pair.
PDL_POE_IRQ_SHORT_67_ENABLE	

<b>Return value</b>	True if all parameters are valid and exclusive; otherwise false.
<b>Category</b>	Port Output Enable
<b>Reference</b>	R_POE_Create
<b>Remarks</b>	<ul style="list-style-type: none"><li>• Call R_POE_Create before using this function.</li><li>• Clearing a level-triggered event flag will fail if the trigger is still asserted.</li><li>• Interrupt disabling is processed at the start of the function and enabling is processed at the end. This allows a flag to be cleared and the interrupt re-enabled in one function call.</li></ul>

<b>Program example</b>	<pre>/* RPDL definitions */ #include "r_pdl_poe.h"  /* RPDL device-specific definitions */ #include "r_pdl_definitions.h"  void func(void) {     /* Select high impedance on the MTU channel 6 and 7 I/O pins */     R_POE_Control(         PDL_POE_MTU67_HI_Z_ON,         PDL_NO_DATA,         PDL_NO_DATA     ); }</pre>
------------------------	--

#### 4) R\_POE\_GetStatus

##### Synopsis

Check the status of the Port Output Enable module.

##### Prototype

```
bool R_POE_GetStatus(
    uint16_t * data    // Status flags pointer
);
```

##### Description

Return the status flags.

##### [data]

The status flags shall be stored in the following format.

b15		b14		b13 – b12		b11		b10		b9		b8	
MTU channel output short detection				0		High impedance request detection							
Channel 6 and 7		Channel 3 and 4				POE11		POE10		-		POE8	
0: Not detected 1: Detected						0: No request 1: Requested							

b7		b6		b5		b4		b3		b2		b1		b0	
High impedance request detection on pin POEn															
-		-		-		POE4		-		-		-		POE0	
0: No request 1: Requested															

##### Return value

True.

##### Category

Port Output Enable

##### Reference

R\_POE\_Control

##### Remarks

- Use R\_POE\_Control to clear the flags.

##### Program example

```
/* RPDL definitions */
#include "r_pdl_poe.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    uint16_t StatusFlags;

    /* Read the POE status */
    R_POE_GetStatus(
        &StatusFlags
    );
}
```

## 4.2.12. General PWM Timer

## 1) R\_GPT\_Set

**Synopsis**

Configure the GPT unit.

**Prototype**

```
bool R_GPT_Set(
    uint32_t data    // Configuration options
);
```

**Description**

Set up the global GPT options.

**[data1]**

Configure the global options. Use “|” to separate each selection.

- Pin selection.

PDL_GPT_PINS_AB_A or PDL_GPT_PINS_AB_B	Select the -A or -B pins for all signals GTIOCxA and GTIOCxB. Not required if these signals are not used.
---	--

- External trigger interrupt control

PDL_GPT_EXT_TRIGGER_INT_DISABLE or PDL_GPT_EXT_TRIGGER_INT_RISING or PDL_GPT_EXT_TRIGGER_INT_FALLING or PDL_GPT_EXT_TRIGGER_INT_BOTH	Disable or enable an interrupt request for rising edge, falling edge or falling and rising edge on pin GTETRQ.
---	---

- Low-speed on-chip oscillator (LOCO) counter control

PDL_GPT_LOCO_COUNT_CLK_ICLK_DIV_1 or PDL_GPT_LOCO_COUNT_CLK_ICLK_DIV_2 or PDL_GPT_LOCO_COUNT_CLK_ICLK_DIV_4 or PDL_GPT_LOCO_COUNT_CLK_ICLK_DIV_8	Select the internal clock signal (ICLK ÷ 1, 2, 4 or 8) for the LOCO counter.
PDL_GPT_LOCO_CLK_DIV_1 or PDL_GPT_LOCO_CLK_DIV_16 or PDL_GPT_LOCO_CLK_DIV_128 or PDL_GPT_LOCO_CLK_DIV_256	Select the LOCO frequency (IWDTCCLK ÷ 1, 16, 128 or 256) to be supplied to the LOCO counter.

- LOCO-derived rising edge skipping control.

PDL_GPT_LOCO_SKIP_NONE or PDL_GPT_LOCO_SKIP_8 or PDL_GPT_LOCO_SKIP_16 or PDL_GPT_LOCO_SKIP_128 or PDL_GPT_LOCO_SKIP_256	Select an interrupt request on every, every 8th, every 16th, every 128th or every 256th rising edge.
PDL_GPT_LOCO_RESULT_SKIP_DISABLE or PDL_GPT_LOCO_RESULT_SKIP_ENABLE	Disable or enable skipping of count results transfers at the same interval as the interrupt request skipping.

- LOCO event interrupt request selection. Each event is disabled by default.

PDL_GPT_LOCO_INT_RISING_ENABLE	LOCO-derived rising edge (using the selected skipping interval).
PDL_GPT_LOCO_INT_DEVIATION_ENABLE	The LOCO frequency deviation has exceeded a permissible limit.
PDL_GPT_LOCO_INT_OVERFLOW_ENABLE	LOCO counter overflow.

- DTC event trigger control.

PDL_GPT_EXT_LOCO_DTC_TRIGGER_DISABLE or PDL_GPT_EXT_LOCO_DTC_TRIGGER_ENABLE	Activate the DTC on a valid external trigger or enabled LOCO event interrupt.
--	---

**Return value**

True if all parameters are valid and exclusive; otherwise false.

**Category**

General PWM Timer unit

**Reference**

R\_GPT\_Create

**Remarks**

- Before calling R\_GPT\_Create, call this function to configure the relevant pins.
- The callback function for external trigger and LOCO event interrupts is specified using function func6 when R\_GPT\_Create is used to configure GPT channel 0.
- The 80- and 64-pin packages do not have GTIOC3A and GTIOC3B pins.

**Program example**

```
/* RPD_L definitions */
#include "r_pdl_gpt.h"

/* RPD_L device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Use the GPT -A pins */
    R_GPT_Set(
        PDL_GPT_PINS_AB_A
    );
}
```

## 2) R\_GPT\_Create

### Synopsis

Configure a GPT channel.

### Prototype

```
bool R_GPT_Create(
    uint8_t data1,           // Channel selection
    R_GPT_Create_structure ptr // A pointer to the structure
);
```

R\_GPT\_Create\_structure members:

```
uint16_t data2 // Control options
uint32_t data3 // Interrupt and data transfer options
uint32_t data4 // Automatic clearing and hardware control options
uint32_t data5 // Hardware control selections
uint32_t data6 // I/O pin control options
uint32_t data7 // I/O pin control options
uint32_t data8 // ADC trigger and skipping control options
uint32_t data9 // Buffer control options
uint32_t data10 // Negate and Dead-time control options
void * func1 // Callback function
void * func2 // Callback function
void * func3 // Callback function
uint8_t data11 // Interrupt priority level
void * func4 // Callback function
void * func5 // Callback function
void * func6 // Callback function
uint8_t data12 // Interrupt priority level
```

### Description (1/7)

Set up a GPT channel.

#### [data1]

The channel number n (where n = 0 to 3).

#### [data2]

Control options. If multiple selections are required, use “|” to separate each selection.

- Operation mode.

PDL_GPT_MODE_SAW or	Saw-wave mode.
PDL_GPT_MODE_SAW_ONE_SHOT or	Saw-wave one-shot pulse mode.
PDL_GPT_MODE_TRIANGLE_1 or PDL_GPT_MODE_TRIANGLE_2 or PDL_GPT_MODE_TRIANGLE_3	Triangle-wave PWM mode 1, 2 or 3.

- Counter clock source selection.

PDL_GPT_CLK_ICLK_DIV_1 or PDL_GPT_CLK_ICLK_DIV_2 or PDL_GPT_CLK_ICLK_DIV_4 or PDL_GPT_CLK_ICLK_DIV_8	The internal clock signal ICLK ÷ 1, 2, 4 or 8.
---	--

**Description (2/7)****[data3]**

Interrupt and data transfer options. If multiple selections are required, use “|” to separate each selection.

The default settings are shown in **bold**.

- Event interrupt request selection.

<b>PDL_GPT_IRQ_A_DISABLE</b> or <b>PDL_GPT_IRQ_A_ENABLE</b>	GTCCRA input capture or compare match.
<b>PDL_GPT_IRQ_B_DISABLE</b> or <b>PDL_GPT_IRQ_B_ENABLE</b>	GTCCRB input capture or compare match.
<b>PDL_GPT_IRQ_C_DISABLE</b> or <b>PDL_GPT_IRQ_C_ENABLE</b>	GTCCRC compare match.
<b>PDL_GPT_IRQ_D_DISABLE</b> or <b>PDL_GPT_IRQ_D_ENABLE</b>	GTCCRD compare match.
<b>PDL_GPT_IRQ_E_DISABLE</b> or <b>PDL_GPT_IRQ_E_ENABLE</b>	GTCCRE compare match.
<b>PDL_GPT_IRQ_F_DISABLE</b> or <b>PDL_GPT_IRQ_F_ENABLE</b>	GTCCRF compare match.
<b>PDL_GPT_IRQ_DEADTIME_DISABLE</b> or <b>PDL_GPT_IRQ_DEADTIME_ENABLE</b>	Dead time error.
<b>PDL_GPT_IRQ_OU_DISABLE</b> or <b>PDL_GPT_IRQ_OU_OVER</b> or <b>PDL_GPT_IRQ_OU_UNDER</b> or <b>PDL_GPT_IRQ_OU_BOTH</b>	Select the Overflow / underflow detection.

- DTC event trigger control.

<b>PDL_GPT_CMICA_DTC_TRIGGER_DISABLE</b> or <b>PDL_GPT_CMICA_DTC_TRIGGER_ENABLE</b>	GTCCRA compare match or input capture.
<b>PDL_GPT_CMICB_DTC_TRIGGER_DISABLE</b> or <b>PDL_GPT_CMICB_DTC_TRIGGER_ENABLE</b>	GTCCRB compare match or input capture.
<b>PDL_GPT_CMCDDE_DTC_TRIGGER_DISABLE</b> or <b>PDL_GPT_CMCDDE_DTC_TRIGGER_ENABLE</b>	GTCCRC or GTCCRD input capture or Dead time error.
<b>PDL_GPT_CMEF_DTC_TRIGGER_DISABLE</b> or <b>PDL_GPT_CMEF_DTC_TRIGGER_ENABLE</b>	GTCCRE or GTCCRF input capture.
<b>PDL_GPT_OU_DTC_TRIGGER_DISABLE</b> or <b>PDL_GPT_OU_DTC_TRIGGER_ENABLE</b>	Counter overflow or underflow.

If a DTC trigger is enabled, remember to enable the appropriate interrupt request.



**Description (3/7)****[data4]**

Automatic clearing and hardware control options.

If multiple selections are required, use “|” to separate each selection.

The default settings are shown in **bold**. Specify PDL\_NO\_DATA to use the defaults.

- Automatic counter clearing.

<b>PDL_GPT_CLEAR_DISABLE</b> or PDL_GPT_CLEAR_A or PDL_GPT_CLEAR_B or	Automatic clearing is disabled.
PDL_GPT_CLEAR_SYNC_CH_0 or PDL_GPT_CLEAR_SYNC_CH_1 or PDL_GPT_CLEAR_SYNC_CH_2 or PDL_GPT_CLEAR_SYNC_CH_3	Cleared by GTCCRA or GTCCRB input capture.
	Clearing at the same time as another channel m (where m ≠ n).

- Hardware counter start, stop and clearing.

<b>PDL_GPT_HW_START_DISABLE</b> or PDL_GPT_HW_START_RISING or PDL_GPT_HW_START_FALLING or PDL_GPT_HW_START_BOTH	Disable or enable start control from another peripheral. If enabled, select a start source using parameter data5.
<b>PDL_GPT_HW_STOP_DISABLE</b> or PDL_GPT_HW_STOP_RISING or PDL_GPT_HW_STOP_FALLING or PDL_GPT_HW_STOP_BOTH	Disable or enable stop control from another peripheral. If enabled, select a stop source using parameter data5.
<b>PDL_GPT_HW_CLEAR_DISABLE</b> or PDL_GPT_HW_CLEAR_RISING or PDL_GPT_HW_CLEAR_FALLING or PDL_GPT_HW_CLEAR_BOTH	Disable or enable counter clearing control from another peripheral. If enabled, select a clear source using parameter data5.

**[data5]**

Hardware control selections.

If multiple selections are required, use “|” to separate each selection.

- Hardware start source selection. Ignored if hardware start control is disabled.

PDL_GPT_HW_START_AN000 or PDL_GPT_HW_START_AN001 or PDL_GPT_HW_START_AN002 or PDL_GPT_HW_START_AN100 or PDL_GPT_HW_START_AN101 or PDL_GPT_HW_START_AN102 or	Comparator detection on 12-bit ADC input ANxxx.
PDL_GPT_HW_START_GTI0C3A_IN or PDL_GPT_HW_START_GTI0C3B_IN or	A valid edge on the GPT pin.
PDL_GPT_HW_START_GTI0C3A_OUT or PDL_GPT_HW_START_GTI0C3B_OUT or	A valid edge on the GPT output compare. Not valid for channel 3.
PDL_GPT_HW_START_GTETR0G	A valid edge on the GTETR0G pin.

- Hardware stop / clear selection. Ignored if both hardware stop and clear control are disabled.

PDL_GPT_HW_STOP_CLEAR_AN000 or PDL_GPT_HW_STOP_CLEAR_AN001 or PDL_GPT_HW_STOP_CLEAR_AN002 or PDL_GPT_HW_STOP_CLEAR_AN100 or PDL_GPT_HW_STOP_CLEAR_AN101 or PDL_GPT_HW_STOP_CLEAR_AN102 or	Comparator detection on 12-bit ADC input ANxxx.
PDL_GPT_HW_STOP_CLEAR_GTI0C3A_IN or PDL_GPT_HW_STOP_CLEAR_GTI0C3B_IN or	A valid edge on the GPT pin.
PDL_GPT_HW_STOP_CLEAR_GTI0C3A_OUT or PDL_GPT_HW_STOP_CLEAR_GTI0C3B_OUT or	A valid edge on the GPT output compare. Not valid for channel 3.
PDL_GPT_HW_STOP_CLEAR_GTETR0G	A valid edge on the GTETR0G pin.

**Description (4/7)****[data6]**

I/O pin control options. If multiple selections are required, use “|” to separate each selection. The default settings are shown in **bold**. Specify PDL\_NO\_DATA to use the defaults.

- Compare match / Input capture selection for pin GTIOCnA

<b>PDL_GPT_A_DISABLED</b> or	Not used, or
PDL_GPT_A_CM_RETAIN or PDL_GPT_A_CM_LOW or PDL_GPT_A_CM_HIGH or PDL_GPT_A_CM_INVERT or	At a compare match the output is retained, set low, set high or toggled, or
PDL_GPT_A_IC_RISING_EDGE or PDL_GPT_A_IC_FALLING_EDGE or PDL_GPT_A_IC_BOTH_EDGES	Input capture at rising edge, falling edge or both edges.

- Additional output settings for pin GTIOCnA. Required only if Compare match is enabled.

PDL_GPT_A_LOW_LOW or PDL_GPT_A_LOW_HIGH or PDL_GPT_A_HIGH_LOW or PDL_GPT_A_HIGH_HIGH or PDL_GPT_A_RETAIN	Set the output states at counter start and stop.
PDL_GPT_A_CYCLE_RETAIN or PDL_GPT_A_CYCLE_LOW or PDL_GPT_A_CYCLE_HIGH or PDL_GPT_A_CYCLE_INVERT	At the timer cycle end the output is retained, set low, set high or toggled.

**[data7]**

I/O pin control options. If multiple selections are required, use “|” to separate each selection. The default settings are shown in **bold**. Specify PDL\_NO\_DATA to use the defaults.

- Compare match / Input capture selection for pin GTIOCnB

<b>PDL_GPT_B_DISABLED</b> or	Not used, or
PDL_GPT_B_CM_RETAIN or PDL_GPT_B_CM_LOW or PDL_GPT_B_CM_HIGH or PDL_GPT_B_CM_INVERT or	At a compare match the output is retained, set low, set high or toggled, or
PDL_GPT_B_IC_RISING_EDGE or PDL_GPT_B_IC_FALLING_EDGE or PDL_GPT_B_IC_BOTH_EDGES	Input capture at rising edge, falling edge or both edges.

- Additional output settings for pin GTIOCnB. Required only if Compare match is enabled.

PDL_GPT_B_LOW_LOW or PDL_GPT_B_LOW_HIGH or PDL_GPT_B_HIGH_LOW or PDL_GPT_B_HIGH_HIGH or PDL_GPT_B_RETAIN	Set the output states at counter start and stop.
PDL_GPT_B_CYCLE_RETAIN or PDL_GPT_B_CYCLE_LOW or PDL_GPT_B_CYCLE_HIGH or PDL_GPT_B_CYCLE_INVERT	At the timer cycle end the output is retained, set low, set high or toggled.

**Description (5/7)****[data8]**

ADC trigger and skipping control options. If multiple selections are required, use “|” to separate each selection.

The default settings are shown in **bold**. Specify PDL\_NO\_DATA to use the defaults.

- ADC conversion trigger selection.

<b>PDL_GPT_ADC_TRIG_A_UP_DISABLE</b> or PDL_GPT_ADC_TRIG_A_UP_ENABLE	Disable or enable ADC start requests on a GTADTRA compare match during up and / or down counting.
<b>PDL_GPT_ADC_TRIG_A_DOWN_DISABLE</b> or PDL_GPT_ADC_TRIG_A_DOWN_ENABLE	
<b>PDL_GPT_ADC_TRIG_B_UP_DISABLE</b> or PDL_GPT_ADC_TRIG_B_UP_ENABLE	Disable or enable ADC start requests on a GTADTRB compare match during up and / or down counting.
<b>PDL_GPT_ADC_TRIG_B_DOWN_DISABLE</b> or PDL_GPT_ADC_TRIG_B_DOWN_ENABLE	

- Interrupt and ADC trigger skipping control.

MicroApT and ADC trigger skipping control.		
PDL_GPT_INT_SKIP_OU_DISABLE or PDL_GPT_INT_SKIP_OU_OVER or PDL_GPT_INT_SKIP_OU_UNDER or PDL_GPT_INT_SKIP_OU_BOTH	Disable skipping, or select skipping for overflow (crest), underflow (trough) or both overflow (crest) and underflow (trough).	
PDL_GPT_INT_SKIP_1 or PDL_GPT_INT_SKIP_2 or PDL_GPT_INT_SKIP_3 or PDL_GPT_INT_SKIP_4 or PDL_GPT_INT_SKIP_5 or PDL_GPT_INT_SKIP_6 or PDL_GPT_INT_SKIP_7	If skipping is enabled, select the skipping count.	
PDL_GPT_INT_SKIP_A	If skipping is enabled, select other events to be skipped.	GTCCRA Compare Match or Input Capture
PDL_GPT_INT_SKIP_B		GTCCRB Compare Match or Input Capture
PDL_GPT_INT_SKIP_C		GTCCRC Compare Match
PDL_GPT_INT_SKIP_D		GTCCRD Compare Match
PDL_GPT_INT_SKIP_E		GTCCRE Compare Match
PDL_GPT_INT_SKIP_F		GTCCRF Compare Match
PDL_GPT_ADC_TRIG_SKIP_A		GTADTRA ADC converter start request
PDL_GPT_ADC_TRIG_SKIP_B		GTADTRB ADC converter start request

**Description (6/7)****[data9]**

Buffer control options. If multiple selections are required, use "[" to separate each selection. The default settings are shown in **bold**. Specify PDL\_NO\_DATA to use the defaults.

- GTCCRA buffer operation

<b>PDL_GPT_BUFFER_CMIC_A_DISABLE</b> or PDL_GPT_BUFFER_CMIC_A_SINGLE or PDL_GPT_BUFFER_CMIC_A_DOUBLE	Disable or select single or double buffer operation for register GTCCRA.
--	--

- GTCCRB buffer operation

<b>PDL_GPT_BUFFER_CMIC_B_DISABLE</b> or PDL_GPT_BUFFER_CMIC_B_SINGLE or PDL_GPT_BUFFER_CMIC_B_DOUBLE	Disable or select single or double buffer operation for register GTCCRB.
--	--

- GTPR buffer operation

<b>PDL_GPT_BUFFER_CYCLE_DISABLE</b> or PDL_GPT_BUFFER_CYCLE_SINGLE or PDL_GPT_BUFFER_CYCLE_DOUBLE	Disable or select single or double buffer operation for register GTPR.
---	--

- GTADTRA buffer operation

<b>PDL_GPT_BUFFER_ADC_TRIG_A_DISABLE</b> or	Disable buffer operation for register GTADTRA or select one of the following:
PDL_GPT_BUFFER_ADC_TRIG_A_CREST or PDL_GPT_BUFFER_ADC_TRIG_A_TROUGH or PDL_GPT_BUFFER_ADC_TRIG_A_BOTH or	Triangle waves: Select transfer at crest, trough or crest and trough.
PDL_GPT_BUFFER_ADC_TRIG_A_SAW	Saw waves: Select transfer at underflow or overflow.

PDL_GPT_BUFFER_ADC_TRIG_A_SINGLE or PDL_GPT_BUFFER_ADC_TRIG_A_DOUBLE	If GTADTRA buffer operation is enabled, select single or double buffer operation.
---	--

- GTADTRB buffer operation

<b>PDL_GPT_BUFFER_ADC_TRIG_B_DISABLE</b> or	Disable buffer operation for register GTADTRB or select one of the following:
PDL_GPT_BUFFER_ADC_TRIG_B_CREST or PDL_GPT_BUFFER_ADC_TRIG_B_TROUGH or PDL_GPT_BUFFER_ADC_TRIG_B_BOTH or	Triangle waves: Select transfer at crest, trough or crest and trough.
PDL_GPT_BUFFER_ADC_TRIG_B_SAW	Saw waves: Select transfer at underflow or overflow.

PDL_GPT_BUFFER_ADC_TRIG_B_SINGLE or PDL_GPT_BUFFER_ADC_TRIG_B_DOUBLE	If GTADTRB buffer operation is enabled, select single or double buffer operation.
---	--

**Description (7/7)****[data10]**

Negate and Dead-time control options.

If multiple selections are required, use “|” to separate each selection.

The default settings are shown in **bold**. Specify PDL\_NO\_DATA to use the defaults.

- Negate control selection

<b>PDL_GPT_NEGATE_A_DISABLE</b> or PDL_GPT_NEGATE_A_LOW or PDL_GPT_NEGATE_A_HIGH	Disable negate operation or set the negate state for pin GTIOCnA.
<b>PDL_GPT_NEGATE_B_DISABLE</b> or PDL_GPT_NEGATE_B_LOW or PDL_GPT_NEGATE_B_HIGH	Disable negate operation or set the negate state for pin GTIOCnB.
PDL_GPT_NEGATE_AN000 or PDL_GPT_NEGATE_AN001 or PDL_GPT_NEGATE_AN002 or PDL_GPT_NEGATE_AN100 or PDL_GPT_NEGATE_AN101 or PDL_GPT_NEGATE_AN102 or PDL_GPT_NEGATE_GTETRG or PDL_GPT_NEGATE_SOFTWARE	If negate operation is enabled, select the negate source.
PDL_GPT_NEGATE_0 or PDL_GPT_NEGATE_1	If negate operation is enabled, select operation when the source becomes 0 or 1.

- Dead time control selection. Ignored in saw-wave PWM mode.

<b>PDL_GPT_DEAD_TIME_DISABLE</b> or PDL_GPT_DEAD_TIME_ENABLE	Disable or enable dead time control.
<b>PDL_GPT_DEAD_TIME_UP_BUFFER_DISABLE</b> or PDL_GPT_DEAD_TIME_UP_BUFFER_ENABLE	Disable or enable buffer operation for up-counting.
<b>PDL_GPT_DEAD_TIME_DOWN_BUFFER_DISABLE</b> or PDL_GPT_DEAD_TIME_DOWN_BUFFER_ENABLE or PDL_GPT_DEAD_TIME_DOWN_MATCH_UP	Disable or enable buffer operation for down-counting, or select loading of the up-counting value.

If any callback function is not required, specify PDL\_NO\_FUNC.

**[func1]**

The function to be called when a Compare match / Input capture A event occurs.

**[func2]**

The function to be called when a Compare match / Input capture B event occurs.

**[func3]**

The function to be called when a Compare match C, Compare match D or Dead Time error event occurs.

**[data11]**

The interrupt priority level for the above events.

Select between 1 (lowest priority) and 15 (highest priority).

This parameter will be ignored if PDL\_NO\_FUNC is specified for all parameters func(1 to 3).

**[func4]**

The function to be called when a Compare match E or Compare match F event occurs.

**[func5]**

The function to be called when an overflow and / or underflow occurs.

**[func6]**

The function to be called when an enabled external trigger or LOCO counter interrupt request occurs. Ignored for n ≠ 0.

**[data12]**

The interrupt priority level for the above events.

Select between 1 (lowest priority) and 15 (highest priority).

This parameter will be ignored if PDL\_NO\_FUNC is specified for all parameters func(4 to 6).

<b>Return value</b>	True if all parameters are valid and exclusive; otherwise false.
<b>Category</b>	General PWM Timer unit
<b>Reference</b>	R_GPT_Set, R_GPT_ControlChannel, R_GPT_ControlUnit
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• If an I/O pin (GTIOCnx, where x = A or B) is made active, this function will configure that pin for input or output and disable other functions on that pin.</li> <li>• If the GTETRQ pin is utilised, this function will configure that pin for input.</li> <li>• The alternative pins are assigned using function R_GPT_Set.</li> <li>• Use R_GPT_ControlChannel to load the registers and start the timer.</li> <li>• If hardware start or stop control is enabled on a channel, starting or stopping <u>any</u> channel using R_GPT_ControlChannel or R_GPT_ControlUnit may cause the channels that use hardware start or stop control to change state in error.</li> <li>• If a callback function is specified, this function will enable the relevant CPU interrupt. Please see the notes on callback function usage in §6.</li> <li>• A callback function is executed by the interrupt processing function. This means that no other interrupt can be processed until the callback function has completed.</li> <li>• A companion function, R_GPT_Create_load_defaults, can be used to load the default values into the structure.</li> <li>• The RX62G supports a PWM edge delay circuit which can be used to control the timing with which signals on the two PWM output pins for each channel rise and fall. This can be configured using functions R_GPT_EdgeDelay_Create and R_GPT_EdgeDelay_Control. It is recommended to enable the edge delay before using this function.</li> </ul>

**Program example**

```

/* RPDL definitions */
#include "r_pdl_gpt.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Allocate a copy of the structure */
    R_GPT_Create_structure ch_parameters;

    /* Load the defaults */
    R_GPT_Create_load_defaults(&ch_parameters);

    /* Set the non-default options for channel 4 */
    ch_parameters.data2 = PDL_GPT_MODE_SAW | PDL_GPT_CLK_ICLK_DIV_1;

    R_GPT_Create(
        2,
        &ch_parameters
    );
}

```

### 3) R\_GPT\_Destroy

**Synopsis**

Disable the GPT unit.

**Prototype**

```
bool R_GPT_Destroy(  
    uint8_t data    // Unit selection  
);
```

**Description**

Shut down the GPT unit and reduce the power consumption.

**[data]**

The unit number n (where n = 0).

**Return value**

True.

**Category**

General PWM Timer unit

**Reference**

None.

**Remarks**

- The unit is put into the stop state.

**Program example**

```
/* RPDL definitions */  
#include "r_pdl_gpt.h"  
  
/* RPDL device-specific definitions */  
#include "r_pdl_definitions.h"  
  
void func(void)  
{  
    /* Shutdown all GPT channels */  
    R_GPT_Destroy(  
        0  
    );  
}
```

## 4) R\_GPT\_ControlChannel

**Synopsis**

Control a GPT channel.

**Prototype**

```
bool R_GPT_ControlChannel(
    uint8_t data1,           // Channel selection
    uint32_t data2,          // Control options
    uint32_t data3,          // Register selection
    R_GPT_ControlChannel_structure_ptr // A pointer to the register structure
                                (Specify PDL_NO_PTR if not required.)
);
```

R\_GPT\_ControlChannel\_structure members:

```
uint16_t data4 // Timer counter register value
uint16_t data5 // General register A value
uint16_t data6 // General register B value
uint16_t data7 // General register C value
uint16_t data8 // General register D value
uint16_t data9 // General register E value
uint16_t data10 // General register F value
uint16_t data11 // Cycle setting register value
uint16_t data12 // Cycle setting buffer register value
uint16_t data13 // Cycle setting double buffer register value
uint16_t data14 // ADC start request A register value
uint16_t data15 // ADC start request A buffer register value
uint16_t data16 // ADC start request A double buffer register value
uint16_t data17 // ADC start request B register value
uint16_t data18 // ADC start request B buffer register value
uint16_t data19 // ADC start request B double buffer register value
uint16_t data20 // Dead-time up-counting register value
uint16_t data21 // Dead-time up-counting buffer register value
uint16_t data22 // Dead-time down-counting register value
uint16_t data23 // Dead-time down-counting buffer register value
```

**Description (1/3)**

Modify a timer channel's registers.

**[data1]**

The channel number n (where n = 0 to 3).

**[data2]**

The channel settings to be modified. All selections are optional.  
If multiple selections are required, use “|” to separate each selection.  
Specify PDL\_NO\_DATA if no change is required.

## • Counter stop / start

PDL_GPT_STOP	Stop the count operation.
PDL_GPT_START	Start the count operation.

## • Counter clearing

PDL_GPT_COUNTER_CLEAR	Clear the counter.
-----------------------	--------------------

## • Buffer operation control

PDL_GPT_BUFFER_CMIC_STOP	Disable and/or enable buffer operation using Compare match or Input capture.
PDL_GPT_BUFFER_CMIC_START	
PDL_GPT_BUFFER_CYCLE_STOP	Disable and/or enable buffer operation using over/under flow.
PDL_GPT_BUFFER_CYCLE_START	
PDL_GPT_BUFFER_ADC_TRIG_STOP	Disable and/or enable buffer operation using ADC start triggers.
PDL_GPT_BUFFER_ADC_TRIG_START	
PDL_GPT_BUFFER_DEAD_TIME_STOP	Disable and/or enable buffer operation using dead time up/down compare match.
PDL_GPT_BUFFER_DEAD_TIME_START	

## • Forced buffer operation

PDL_GPT_BUFFER_CMIC_FORCE	Force a buffer transfer of registers GTCCRM and GTCCRB.
---------------------------	---



**Description (2/3)**

• Write protection	
PDL_GPT_WRITE_ENABLE	Allow writing to the channel registers.
PDL_GPT_WRITE_DISABLE	Prevent writing to the channel registers.
• Count direction	
PDL_GPT_COUNT_DIRECTION_DOWN or PDL_GPT_COUNT_DIRECTION_UP	Set the count direction.
PDL_GPT_COUNT_DIRECTION_FORCE	Forcibly set the count direction.
• Software negate control	
PDL_GPT_NEGATE_ON or PDL_GPT_NEGATE_OFF	If software negate control has been enabled (in parameter data10 of R_GPT_Create), control the negate state.
• Output protection temporary release control	
PDL_GPT_OUTPUT_PROTECTION_RELEASE or PDL_GPT_OUTPUT_PROTECTION_RESTORE	Release or restore the protected state of the GTIOCnB pin

**[data3]**

The registers to be modified. All selections are optional.  
If multiple selections are required, use “|” to separate each selection.  
Specify PDL\_NO\_DATA if no change is required.

• The registers to be modified.	
PDL_GPT_REGISTER_COUNTER	Timer counter.
PDL_GPT_REGISTER_A	General register A.
PDL_GPT_REGISTER_B	General register B.
PDL_GPT_REGISTER_C	General register C.
PDL_GPT_REGISTER_D	General register D.
PDL_GPT_REGISTER_E	General register E.
PDL_GPT_REGISTER_F	General register F.
PDL_GPT_REGISTER_CYCLE	Cycle setting register.
PDL_GPT_REGISTER_CYCLE_BUFFER	Cycle setting buffer register.
PDL_GPT_REGISTER_CYCLE_DOUBLE	Cycle setting double buffer register.
PDL_GPT_REGISTER_ADC_TRIG_A	ADC start request register.
PDL_GPT_REGISTER_ADC_TRIG_A_BUFFER	ADC start request buffer register.
PDL_GPT_REGISTER_ADC_TRIG_A_DOUBLE	ADC start request double buffer register.
PDL_GPT_REGISTER_ADC_TRIG_B	ADC start request register.
PDL_GPT_REGISTER_ADC_TRIG_B_BUFFER	ADC start request buffer register.
PDL_GPT_REGISTER_ADC_TRIG_B_DOUBLE	ADC start request double buffer register.
PDL_GPT_REGISTER_DEAD_TIME_UP	Dead-time up-counting register.
PDL_GPT_REGISTER_DEAD_TIME_UP_BUFFER	Dead-time up-counting buffer register.
PDL_GPT_REGISTER_DEAD_TIME_DOWN	Dead-time down-counting register.
PDL_GPT_REGISTER_DEAD_TIME_DOWN_BUFFER	Dead-time down-counting buffer register.

The following register values will be ignored if they have not been selected above.

**[data4]**

The timer counter (GTCNT) value.

**[data5]**

The general register A (GTCCRA) value.

**[data6]**

The general register B (GTCCRB) value.

**[data7]**

The general register C (GTCCRC) value.

**[data8]**

The general register D (GTCCRD) value.

<b>Description (3/3)</b>	<p><b>[data9]</b> The general register E (GTCCRE) value.</p> <p><b>[data10]</b> The general register F (GTCCRF) value.</p> <p><b>[data11]</b> The cycle setting register (GTPR) value.</p> <p><b>[data12]</b> The cycle setting buffer register (GTPBR) value.</p> <p><b>[data13]</b> The cycle setting double buffer register (GTPDBR) value.</p> <p><b>[data14]</b> The ADC start request register (GTADTRA) value.</p> <p><b>[data15]</b> The ADC start request buffer register (GTADTBRA) value.</p> <p><b>[data16]</b> The ADC start request double buffer register (GTADTDBRA) value.</p> <p><b>[data17]</b> The ADC start request register (GTADTRB) value.</p> <p><b>[data18]</b> The ADC start request buffer register (GTADTBRB) value.</p> <p><b>[data19]</b> The ADC start request double buffer register (GTADTDBRB) value.</p> <p><b>[data20]</b> The dead-time up-counting register (GTDVU) value.</p> <p><b>[data21]</b> The dead-time up-counting buffer register (GTDBU) value.</p> <p><b>[data22]</b> The dead-time down-counting register (GTDVD) value.</p> <p><b>[data23]</b> The dead-time down-counting buffer register (GTDBD) value.</p>
<b>Return value</b>	True if all parameters are valid and exclusive; otherwise false.
<b>Category</b>	General PWM Timer unit
<b>Reference</b>	R_GPT_Create
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• The Stop operation is executed at the start of this function. The Start operation is executed at the end. Both options can be selected together with other changes in one function call.</li> <li>• The buffer operation disable control is executed at the start of this function. The buffer operation enable control is executed at the end. Both options can be selected together with buffer register access in one function call.</li> <li>• If Stop or Start control is selected, any channel that has hardware start or stop control enabled may change state in error.</li> <li>• The LOCO counter Stop operation is executed at the start of this function. The Start operation is executed at the end. Both options can be selected together with other changes in one function call.</li> </ul>

**Program example**

```
/* RPDL definitions */
#include "r_pdl_gpt.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Allocate a copy of the structure for the selected channel */
    R_GPT_ControlChannel_structure gpt_1_control_parameters;

    /* Set the register values for channel 1 */
    gpt_1_control_parameters.data4 = 0xFFDD;
    gpt_1_control_parameters.data6 = 0x0020;

    /* Load the register values and start channel 1 */
    R_GPT_ControlChannel(
        1,
        PDL_GPT_STOP | PDL_GPT_START,
        PDL_GPT_REGISTER_COUNTER | PDL_GPT_REGISTER_B,
        &gpt_1_control_parameters
    );
}
```

## 5) R\_GPT\_ControlUnit

### Synopsis

Control the GPT unit.

### Prototype

```
bool R_GPT_ControlUnit(
    uint8_t data1,    // Unit selection
    uint32_t data2,    // Control options
    uint16_t data3,    // Clock positive tolerance
    uint16_t data4     // Clock negative tolerance
);
```

### Description (1/2)

Modify the timer unit registers.

#### [data1]

The unit number n (where n = 0).

#### [data2]

The unit settings to be modified.

If multiple selections are required, use “|” to separate each selection.

Specify PDL\_NO\_DATA if no change is required.

#### • Counter stop

PDL_GPT_STOP_CH_0	Stop the count operation for the selected channels. The stop operation will be simultaneous.
PDL_GPT_STOP_CH_1	
PDL_GPT_STOP_CH_2	
PDL_GPT_STOP_CH_3	

#### • Counter start

PDL_GPT_START_CH_0	Start the count operation for the selected channels. The start operation will be simultaneous.
PDL_GPT_START_CH_1	
PDL_GPT_START_CH_2	
PDL_GPT_START_CH_3	

#### • Counter clearing

PDL_GPT_CLEAR_CH_0	Clear the counters for the selected channels. The clear operation will be simultaneous.
PDL_GPT_CLEAR_CH_1	
PDL_GPT_CLEAR_CH_2	
PDL_GPT_CLEAR_CH_3	

#### • LOCO counter stop / start

PDL_GPT_LOCO_COUNT_DISABLE	Stop the LOCO count operation.
PDL_GPT_LOCO_COUNT_ENABLE	Start the LOCO count operation.

#### • LOCO counter clearing

PDL_GPT_LOCO_COUNT_CLEAR	Clear the LOCO counter (LCNT) to 0.
--------------------------	-------------------------------------

#### • LOCO count result initialisation

PDL_GPT_LOCO_RESULTS_INIT	Initialise the LOCO count result registers (LCNT01 to LCNT15) using the first count (LCNT00) value.
---------------------------	---

#### • LOCO count deviation update

PDL_GPT_LOCO_DEVIATION_TOLERANCE or	Parameters data3 and data4 contain the positive and negative tolerances.
PDL_GPT_LOCO_DEVIATION_REGISTER	Parameters data3 and data4 contain the register values.

#### [data3]

If required, either the maximum positive deviation for the main clock (ICLK) as a percentage, or the LCNTDU register value.

If the LOCO count deviation update is not required, specify PDL\_NO\_DATA.

<b>Description (2/2)</b>	<p><b>[data4]</b>          If required, either the maximum negative deviation for the main clock (ICLK) as a percentage, or the LCNTDL register value.          If the LOCO count deviation update is not required, specify PDL_NO_DATA.</p>
<b>Return value</b>	True if all parameters are valid and exclusive; otherwise false.
<b>Category</b>	General PWM Timer unit
<b>Reference</b>	R_GPT_Set, R_IWDT_Set, R_IWDT_Control
<b>Remarks</b>	<ul style="list-style-type: none"> <li>The Stop operations are executed at the start of this function. The Start operations are executed at the end. Both options can be selected together with other changes in one function call.</li> <li>If Stop or Start control is selected, any channel that has hardware start or stop control enabled may change state in error.</li> <li>If the LOCO counter is enabled, the Independent Watchdog Timer (IWDT) should also be enabled. The following sequence is recommended:               <ol style="list-style-type: none"> <li>Use R_GPT_Set to configure the LOCO counter.</li> <li>Use R_IWDT_Set and R_IWDT_Control to configure and start the IWDT. If the LOCO frequency division is 16 or more, avoid selecting PDL_IWDT_CLOCK_OCO_1.</li> <li>Use this function to initialise and start the LOCO counter.</li> </ol> </li> </ul>

**Program example**

```

/* RPDL definitions */
#include "r_pdl_gpt.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Stop, clear and re-start channels 0 and 1 simultaneously */
    R_GPT_ControlUnit(
        0,
        PDL_GPT_STOP_CH_0 | PDL_GPT_STOP_CH_1 | \
        PDL_GPT_CLEAR_CH_0 | PDL_GPT_CLEAR_CH_1 | \
        PDL_GPT_START_CH_0 | PDL_GPT_START_CH_1,
        PDL_NO_DATA,
        PDL_NO_DATA
    );

    /* Update the LOCO count deviation limits to +10% & -20% */
    R_GPT_ControlUnit(
        0,
        PDL_GPT_LOCO_DEVIATION_TOLERANCE,
        10,
        20
    );
}

```

## 6) R\_GPT\_ReadChannel

### Synopsis

Read from GPT channel registers.

### Prototype

```
bool R_GPT_ReadChannel(
    uint8_t data1,      // Channel selection
    uint16_t * data2,   // Flag storage location
    uint16_t * data3,   // Timer counter register storage location
    uint16_t * data4,   // General register A storage location
    uint16_t * data5,   // General register B storage location
    uint16_t * data6,   // General register C storage location
    uint16_t * data7,   // General register D storage location
    uint16_t * data8,   // General register E storage location
    uint16_t * data9,   // General register F storage location
    uint16_t * data10,  // Cycle setting register storage location
    uint16_t * data11,  // Cycle setting buffer register storage location
    uint16_t * data12,  // Cycle setting double buffer register storage location
    uint16_t * data13,  // ADC start request A register storage location
    uint16_t * data14,  // ADC start request A buffer register storage location
    uint16_t * data15,  // ADC start request A double buffer register storage location
    uint16_t * data16,  // ADC start request B register storage location
    uint16_t * data17,  // ADC start request B buffer register storage location
    uint16_t * data18,  // ADC start request B double buffer register storage location
    uint16_t * data19,  // Dead-time up-counting register storage location
    uint16_t * data20,  // Dead-time up-counting buffer register storage location
    uint16_t * data21,  // Dead-time down-counting register storage location
    uint16_t * data22   // Dead-time down-counting buffer register storage location
);
```

### Description (1/2)

Read any of the timer's counter, compare or status flag registers.

#### [data1]

The channel number n (where n = 0 to 3).

#### [data2]

The status flags shall be stored in the format below.

Specify PDL\_NO\_PTR if the flags are not to be read.

b15		b14		b13 - b12				b11		b10 - b8	
Timer counter status		Output protected state				Dead-time error detection		Interrupt skipping counter			
0: Down 1: Up	0: Stopped 1: Counting	00b: Normal Other: Protected, due to the GTCCRA value at buffer transfer: 01b: GTCCRA = 0 10b: GTCCRA ≥ GTPR at trough 11b: GTCCRA ≥ GTPR at crest				0: Not detected 1: Detected					

b7		b6		b5		b4		b3		b2		b1		b0	
Event detection															
Underflow	Overflow	Compare match								Input capture or compare match					
		F		E		D		C		B			A		
0: Not detected 1: Detected															

If any register value is not required, specify PDL\_NO\_PTR.

#### [data3]

A pointer to where the Timer counter register value shall be stored.

#### [data4]

A pointer to where the General register A value shall be stored.

#### [data5]

A pointer to where the General register B value shall be stored.

<b>Description (2/2)</b>	<p><b>[data6]</b> A pointer to where the General register C value shall be stored.</p> <p><b>[data7]</b> A pointer to where the General register D value shall be stored.</p> <p><b>[data8]</b> A pointer to where the General register E value shall be stored.</p> <p><b>[data9]</b> A pointer to where the General register F value shall be stored.</p> <p><b>[data10]</b> A pointer to where the Cycle setting register value shall be stored.</p> <p><b>[data11]</b> A pointer to where the Cycle setting buffer register value shall be stored.</p> <p><b>[data12]</b> A pointer to where the Cycle setting double buffer register value shall be stored.</p> <p><b>[data13]</b> A pointer to where the ADC start request A register value shall be stored.</p> <p><b>[data14]</b> A pointer to where the ADC start request A buffer register value shall be stored.</p> <p><b>[data15]</b> A pointer to where the ADC start request A double buffer register value shall be stored.</p> <p><b>[data16]</b> A pointer to where the ADC start request B register value shall be stored.</p> <p><b>[data17]</b> A pointer to where the ADC start request B buffer register value shall be stored.</p> <p><b>[data18]</b> A pointer to where the ADC start request B double buffer register value shall be stored.</p> <p><b>[data19]</b> A pointer to where the Dead-time up-counting register value shall be stored.</p> <p><b>[data20]</b> A pointer to where the Dead-time up-counting buffer register value shall be stored.</p> <p><b>[data21]</b> A pointer to where the Dead-time down-counting register value shall be stored.</p> <p><b>[data22]</b> A pointer to where the Dead-time down-counting buffer register value shall be stored.</p>
<b>Return value</b>	True.
<b>Category</b>	General PWM Timer unit
<b>Reference</b>	None.
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• If the flags are read, any detection flag that has been set to 1 shall be automatically cleared to 0 by this function.</li> </ul>

## Program example

[illegible]



## 7) R\_GPT\_ReadUnit

### Synopsis

Read the GPT unit flags.

### Prototype

```
bool R_GPT_ReadUnit(
    uint8_t data1,    // Unit selection
    uint8_t * data2,  // The flag storage location
    uint16_t * data3, // The LOCO counter storage location
    uint16_t * data4, // The LOCO count result average storage location
    uint16_t * data5, // The LOCO count upper permissible deviation location
    uint16_t * data6, // The LOCO count lower permissible deviation location
    uint16_t * data7  // The LOCO count result storage location
);
```

### Description

Read the GPT unit status and data

#### [data1]

The unit number n (where n = 0).

#### [data2]

The detection flags shall be stored in the format below.

Specify PDL\_NO\_PTR if the flags are not to be read.

b7 – b5	b4	b3	b2	b1	b0
0	External trigger edge		LOCO counter		
	Falling	Rising	Overflow	Deviation limit	LOCO-derived rising edge
	0: Not detected 1: Detected				

If any register value is not required, specify PDL\_NO\_PTR.

#### [data3]

A pointer to where the LOCO counter (LCNT) register value shall be stored.

#### [data4]

A pointer to where the LOCO count result average (LCNTA) register value shall be stored.

#### [data5]

A pointer to where the LOCO count upper permissible deviation (LCNTDU) register value shall be stored.

#### [data6]

A pointer to where the LOCO count lower permissible deviation (LCNTDL) register value shall be stored.

#### [data7]

A pointer to where the LOCO count result (LCNTxx) registers value shall be stored.  
Provide space for 16 sixteen-bit values.

### Return value

True.

### Category

General PWM Timer unit

### Reference

None.

### Remarks

- If the flags are read, any flag that has been set to 1 shall be automatically cleared to 0 by this function.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_gpt.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

uint8_t Flags;
uint16_t loco_counter;
uint16_t loco_counter_result_average;
uint16_t loco_counter_upper_limit;
uint16_t loco_counter_lower_limit;
uint16_t loco_counter_results[16];

void func(void)
{
    /* Read the status flags and LOCO count registers */
    R_GPT_ReadUnit(
        0,
        &Flags,
        &loco_counter,
        &loco_counter_result_average,
        &loco_counter_upper_limit,
        &loco_counter_lower_limit,
        loco_counter_results
    );
}
```

## 8) R\_GPT\_EdgeDelay\_Create

**Synopsis**

Enable the PWM Edge Delay circuit.

**Prototype**

```
bool R_GPT_EdgeDelay_Create(
    uint8 data1,    // Channel 0 configuration
    uint8 data2,    // Channel 1 configuration
    uint8 data3,    // Channel 2 configuration
    uint8 data4     // Channel 3 configuration
);
```

**Description**

Enable the PWM Edge Delay circuit.

**[data1]**

Channel 0 configuration. Specify PDL\_NO\_DATA to use the default.

- Configuration

PDL_GPT_PWM_DELAY_LEAVE	Leave the current configuration.
PDL_GPT_PWM_DELAY_ENABLE	Enable and activate the delay circuit.
PDL_GPT_PWM_DELAY_DISABLE	Disable the delay circuit.

**[data2]**

Channel 1 configuration. See [data1] for format.

**[data3]**

Channel 2 configuration. See [data1] for format.

**[data4]**

Channel 3 configuration. See [data1] for format.

**Return value**

True if all parameters are valid and exclusive; otherwise false.

**Category**

General PWM Timer unit

**Reference**

None.

**Remarks**

- This function is supported on the RX62G only.
- During execution of this function the ICLK frequency is temporarily halved in value. Therefore, it is recommended to call this function before enabling other modules that rely on ICLK.
- If the GPT module is in the stopped state, the stop state will be cancelled.
- If a delay is going to be enabled for a channel then the counter for that channel will be stopped.
- Function R\_CGC\_Set must be called before any use of this function.
- ICLK must be  $\geq 80\text{MHz}$  to enable the Delay circuit. This function will return false if this is not the case.
- The initialisation process of the delay circuit includes a waiting time that this function automatically provides. If this function is used to enable all of the required delay channels at the same time then this delay period is shared, thus minimising the overall initialisation time.
- When a channel is first enabled this function will set the delay time to zero (no delay). Use function R\_GPT\_EdgeDelay\_Control to set the required delay time.
- It is recommended to use this function before using R\_GPT\_Create.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_gpt.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Enable the PWM delay circuit for channels 0 and 2. */
    R_GPT_EdgeDelay_Create(
        PDL_GPT_PWM_DELAY_ENABLE,
        PDL_GPT_PWM_DELAY_DISABLE,
        PDL_GPT_PWM_DELAY_ENABLE,
        PDL_GPT_PWM_DELAY_DISABLE
    );
}
```

## 9) R\_GPT\_EdgeDelay\_Control

### Synopsis

Control the PWM Edge Delay circuit

### Prototype

```
bool R_GPT_EdgeDelay_Control(
    uint8_t data1,
    uint8_t data2,
    R_GPT_EdgeDelay_Times_structure *    // A pointer to a structure
);
```

R\_GPT\_EdgeDelay\_Times\_structure members:

```
uint8_t GTIOCA_Rising_Delay    // GTIOCA Rising Edge Delay Time
uint8_t GTIOCA_Falling_Delay   // GTIOCA Falling Edge Delay Time
uint8_t GTIOCB_Rising_Delay    // GTIOCB Rising Edge Delay Time
uint8_t GTIOCB_Falling_Delay   // GTIOCB Falling Edge Delay Time
```

### Description

Control the PWM Edge Delay circuit including setting / adjusting the delay times.

#### [data1]

The channel number n (where n = 0 to 3).

#### [data2]

Control option. To set multiple options at the same time, use “|” to separate each value.

- Time adjustment.

PDL_GPT_PWM_DELAY_TIME	Update the delay times with those specified in [data3] to [data6].
------------------------	--

- Activate (this is not required for normal use but provides control of GTDLYCR.DLYEN).

PDL_GPT_PWM_DELAY_ACTIVE or	Activate the delay circuit.
PDL_GPT_PWM_DELAY_INACTIVE	Inactivate (bypass) the delay circuit.

- Reset (This is not required for normal use but provides control of GTDLYCR.DLYRST)

PDL_GPT_PWM_DELAY_RESET	Perform a reset by setting the DLYRST bit high then low.
-------------------------	--

#### [GTIOCA\_Rising\_Delay]

The delay to be applied to the rising edge of GTIOCA.

The delay is specified in fractions of the ICLK period:

0 = no delay.

1 = 1/32 of ICLK.

2 = 2/32 of ICLK.

...

30 = 30/32 of ICLK

31 = 31/32 of ICLK.

#### [GTIOCA\_Falling\_Delay]

The delay to be applied to the falling edge of GTIOCA (See [data3] for format).

#### [GTIOCB\_Rising\_Delay]

The delay to be applied to the rising edge of GTIOCB (See [data3] for format).

#### [GTIOCB\_Falling\_Delay]

The delay to be applied to the falling edge of GTIOCB (See [data3] for format).

### Return value

True if all parameters are valid and exclusive; otherwise false.

### Category

General PWM Timer unit

### Reference

None.

**Remarks**

- This function is supported on the RX62G only.
- Call R\_GPT\_EdgeDelay\_Create to enable a channel before using this function.
- If a delay time adjustment is made the actual adjustment will take place on overflows (in up-counting) or underflows (in down-counting) for saw waves, and in the troughs of triangle waves.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_gpt.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

R_GPT_EdgeDelay_Times_structure Delay2;

void func(void)
{
    /* Adjust channel 2 delay times */
    Delay2.GTIOCA_Rising_Delay = 1;
    Delay2.GTIOCA_Falling_Delay = 2;
    Delay2.GTIOCB_Rising_Delay = 3;
    Delay2.GTIOCB_Falling_Delay = 4;

    R_GPT_EdgeDelay_Control(
        2,
        PDL_GPT_PWM_DELAY_TIME,
        &Delay2
    );
}
```

10) **R\_GPT\_EdgeDelay\_Destroy****Synopsis**

Disable the Edge Delay circuit.

**Prototype**

```
bool R_GPT_EdgeDelay_Destroy(
    uint8_t data    // Unit selection
);
```

**Description**

Disable the edge delay circuit on all channels.

**[data]**

The unit number n (where n = 0).

**Return value**

True.

**Category**

General PWM Timer unit

**Reference**

None.

**Remarks**

- This function is supported on the RX62G only.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_gpt.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Disable the Edge Delay circuits */
    R_GPT_EdgeDelay_Destroy(
        0
    );
}
```

## 4.2.13. Compare Match Timer

## 1) R\_CMT\_Create

**Synopsis**

Configure a CMT channel.

**Prototype**

```
bool R_CMT_Create(
    uint8_t data1, // Timer channel selection
    uint16_t data2, // Configuration selection
    float data3, // Period, frequency or register data
    void * func, // Callback function
    uint8_t data4 // Interrupt priority level
);
```

**Description**

Set up a Compare Match Timer channel and start the timer.

**[data1]**

The channel number n (where n = 0, 1, 2 or 3).

**[data2]**Configure the timer. To set multiple options at the same time, use "|" to separate each value. The default settings are shown in **bold**.

- Clock calculation

PDL_CMT_PERIOD or	The parameter data3 will specify the timer period. The counter clock source and compare match value will be calculated by this function.
PDL_CMT_FREQUENCY or	The parameter data3 will specify the timer frequency. The counter clock source and compare match value will be calculated by this function.
PDL_CMT_PCLK_DIV_8 or PDL_CMT_PCLK_DIV_32 or PDL_CMT_PCLK_DIV_128 or PDL_CMT_PCLK_DIV_512	Select the internal clock signal PCLK ÷ 8, 32, 128 or 512 as the counter clock source. The parameter data3 will be the register CMCOR value.

- DTC trigger control

<b>PDL_CMT_DTC_TRIGGER_DISABLE</b> or <b>PDL_CMT_DTC_TRIGGER_ENABLE</b>	Disable or enable activation of the DTC when a compare match occurs.
--	--

**[data3]**

The data to be used for the register value calculations.

Data use

The timer period in seconds or

The timer frequency in Hz or

The value to be put in register CMCOR

Parameter type

float

float

uint16\_t

**[func]**

The function to be called at the periodic interval. Specify PDL\_NO\_FUNC if not required.

**[data4]**

The interrupt priority level. Select between 1 (lowest priority) and 15 (highest priority). This parameter will be ignored if PDL\_NO\_FUNC is specified for parameter func.

**Return value**

True if all parameters are valid and exclusive; otherwise false.

**Category**

Compare Match Timer

**Reference**

R\_CMT\_Destroy



**Remarks**

- Function R\_CGC\_Set must be called before any use of this function.
- If a callback function is specified, this function will enable the relevant interrupt. Please see the notes on callback function use in §6.
- A callback function is executed by the interrupt processing function. This means that no other interrupt can be processed until the callback function has completed.
- Ensure that the timer channel is stopped before calling this function.
- The timing limits depend on the frequency of the peripheral module clock, PCLK.

	Equation	f <sub>PCLK</sub> (MHz)					
		50	48	12.5	12	32	8
Period <sub>MIN</sub>	$\frac{8}{f_{PCLK}}$	160 ns	166.7 ns	640 ns	666.7 ns	250 ns	1.0 μs
Period <sub>MAX</sub>	$\frac{2^{25}}{f_{PCLK}}$	671 ms	699 ms	2.68s	2.79 s	1.05s	4.19s
f <sub>MAX</sub>	$\frac{f_{PCLK}}{8}$	6.25 MHz	6 MHz	1.56 MHz	1.5 MHz	4.0 MHz	1.0 MHz
f <sub>MIN</sub>	$\frac{f_{PCLK}}{2^{25}}$	1.49 Hz	1.43 Hz	0.37 Hz	0.357 Hz	0.95 Hz	0.24 Hz

- If the requested period is not a multiple of the minimum period, the actual time period will be more than the requested time period.

**Program example**

```

/* RPDL definitions */
#include "r_pdl_cmt.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Configure CMT channel 0 for 10μs operation */
    R_CMT_Create(
        0,
        PDL_CMT_PERIOD,
        10E-6,
        PDL_NO_FUNC,
        0
    );

    /* Configure CMT channel 1 for 1kHz operation */
    R_CMT_Create(
        1,
        PDL_CMT_FREQUENCY,
        1E3,
        PDL_NO_FUNC,
        0
    );

    /* Configure CMT channel 2 using register values */
    R_CMT_Create(
        2,
        PDL_CMT_PCLK_DIV_32,
        0x55AA,
        PDL_NO_FUNC,
        0
    );
}

```

## 2) R\_CMT\_CreateOneShot

### Synopsis

Configure a CMT channel as a one-shot event.

### Prototype

```
bool R_CMT_CreateOneShot(
    uint8_t data1,    // Timer channel selection
    uint16_t data2,   // Configuration selection
    float data3,      // Period
    void * func,      // Callback function
    uint8_t data4     // Interrupt priority level
);
```

### Description

Set up a Compare Match Timer channel and start the timer.

#### [data1]

The channel number *n* (where *n* = 0, 1, 2 or 3).

#### [data2]

Configure the timer.

The default settings are shown in **bold**. Specify PDL\_NO\_DATA to use the defaults.

- Control the CPU during the one-shot operation.

<b>PDL_CMT_CPU_ON</b> or	Allow the CPU to run normally while the one-shot operates.
PDL_CMT_CPU_OFF	Stop the CPU when the one-shot timer starts. The CPU will re-start when any valid interrupt occurs.

- DTC trigger control

<b>PDL_CMT_DTC_TRIGGER_DISABLE</b> or PDL_CMT_DTC_TRIGGER_ENABLE	Disable or enable activation of the DTC when a compare match occurs.
---	---

#### [data3]

The one-shot time period (in seconds).

#### [func]

The function to be called when the one-shot period ends.

If you specify PDL\_NO\_FUNC, this function will wait for the timer to complete before returning. You should always specify a function if PDL\_CMT\_CPU\_OFF is selected to ensure that an interrupt will re-start the CPU.

#### [data4]

The interrupt priority level. Select between 1 (lowest priority) and 15 (highest priority). This parameter will be ignored if PDL\_NO\_FUNC is specified for parameter func.

### Return value

True if all parameters are valid and exclusive; otherwise false.

### Category

Compare Match Timer

### Reference

R\_CGC\_Set

**Remarks**

- Function R\_CGC\_Set must be called before any use of this function.
- Function R\_CMT\_Create is not required.
- Ensure that the timer channel is stopped before calling this function. Note that the timer is stopped automatically when the one-shot period is reached.
- If a callback function is specified, this function will enable the relevant interrupt. Please see the notes on callback function use in §6.
- A callback function is executed by the interrupt processing function. This means that no other interrupt can be processed until the callback function has completed.
- The timing limits depend on the peripheral module clock, PCLK.

	Equation	f <sub>PCLK</sub> (MHz)					
		50	48	12.5	12	32	8
T <sub>MIN</sub>	$\frac{8}{f_{PCLK}}$	160 ns	166.7 ns	640 ns	666.7 ns	250 ns	1 μs
T <sub>MAX</sub>	$\frac{2^{25}}{f_{PCLK}}$	671 ms	699 ms	2.68s	2.79s	1.05s	4.19s

- If the requested period is not a multiple of the minimum period, the actual time period will be more than the requested time period.

**Program example**

```

/* RPDL definitions */
#include "r_pdl_cmt.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Use CMT channel 0 for a 1ms pause */
    R_CMT_CreateOneShot(
        0,
        0,
        1E-3,
        PDL_NO_FUNC,
        0
    );
}

```

### 3) R\_CMT\_Destroy

**Synopsis**

Disable a CMT unit.

**Prototype**

```
bool R_CMT_Destroy(
    uint8_t data // Unit selection
);
```

**Description**

Shut down a CMT unit.

**[data]**

The timer unit n (where n = 0 or 1).  
Unit 0 comprises channels CMT0 and CMT1.  
Unit 1 comprises channels CMT2 and CMT3.

**Return value**

True if the unit selection is valid; otherwise false.

**Category**

Compare Match Timer

**Reference**

R\_CMT\_Create

**Remarks**

- The timer unit is put into the stop state to reduce power consumption.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_cmt.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Shutdown channels 0 and 1 */
    R_CMT_Destroy(
        0
    );
}
```

## 4) R\_CMT\_Control

### Synopsis

Control CMT operation.

### Prototype

```
bool R_CMT_Control(
    uint8_t data1,    // Channel selection
    uint16_t data2,   // Configuration selection
    float data3       // Period, frequency or register data
);
```

### Description

Modify the operation of a CMT channel.

#### [data1]

The channel number *n* (where *n* = 0, 1, 2 or 3).

#### [data2]

Configure the timer channel. To set multiple options at the same time, use “|” to separate each value.

- Counter stop / re-start

PDL_CMT_STOP	Disable the counter clock source.
PDL_CMT_START	Enable the counter clock source.

- Value change request

PDL_CMT_PERIOD or PDL_CMT_FREQUENCY or PDL_CMT_CONSTANT or PDL_CMT_COUNTER	The parameter data3 will contain the new period, frequency, constant register (CMCOT) or counter register (CMCNT) value.
---	--

#### [data3]

The new period, frequency or register value. This will be ignored if a value change is not requested.

Data use	Parameter type
The timer period in seconds or	float
The timer frequency in Hz or	float
The value to be put in the selected register	uint16_t

### Return value

True if all parameters are valid and exclusive; otherwise false.

### Category

Compare Match Timer

### Reference

R\_CMT\_Create

### Remarks

- R\_CMT\_Create must be first be used to configure the channel.
- The Stop operation is executed at the start of this function.  
The Start operation is executed at the end.  
Therefore, both options can be selected together with a value change in one function call.

### Program example

```
/* RPDL definitions */
#include "r_pdl_cmt.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Change channel 2 to 1ms period */
    R_CMT_Control(
        2,
        PDL_CMT_PERIOD,
        1E-3
    );
}
```

## 5) R\_CMT\_Read

### Synopsis

Read CMT channel status and registers.

### Prototype

```
bool R_CMT_Read(
    uint8_t data1,    // Channel selection
    uint8_t * data2,  // A pointer to the data storage location
    uint16_t * data3  // A pointer to the data storage location
);
```

### Description

Read and store the counter value and status flag.

#### [data1]

The channel number *n* (where *n* = 0, 1, 2 or 3).

#### [data2]

The compare match status flag shall be stored in the following format.  
Specify PDL\_NO\_PTR if the flag is not to be read.

b7 – b1		b0
0		0: Idle 1: Compare match condition detected

#### [data3]

A pointer to where the counter value shall be stored. Specify PDL\_NO\_PTR if it is not required.

### Return value

True if all parameters are valid; otherwise false.

### Category

Compare Match Timer

### Reference

R\_CMT\_Create

### Remarks

- If the flag is read and is set to 1, it shall be automatically cleared to 0 by this function.

### Program example

```
/* RPDL definitions */
#include "r_pdl_cmt.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

uint8_t Flags;
uint16_t Counter;

void func(void)
{
    /* Read the channel 2 values */
    R_CMT_Read(
        2,
        &Flags,
        &Counter
    );
}
```

## 4.2.14. Watchdog Timer

## 1) R\_WDT\_Create

**Synopsis**

Configure the Watchdog timer.

**Prototype**

```
bool R_WDT_Create(
    uint16_t data1, // Configuration selection
    void * func,    // Callback function
    uint8_t data2   // Interrupt priority level
);
```

**Description**

Set up and start the Watchdog timer.

**[data1]**

Configure the timer. To set multiple options at the same time, use “|” to separate each value. The default settings are shown in **bold**.

## • Clock selection

PDL\_WDT\_PCLK\_DIV\_4 or  
PDL\_WDT\_PCLK\_DIV\_64 or  
PDL\_WDT\_PCLK\_DIV\_128 or  
PDL\_WDT\_PCLK\_DIV\_512 or  
PDL\_WDT\_PCLK\_DIV\_2048 or  
PDL\_WDT\_PCLK\_DIV\_8192 or  
PDL\_WDT\_PCLK\_DIV\_32768 or  
PDL\_WDT\_PCLK\_DIV\_131072

The division ratio for the internal clock signal PCLK.

## • MCU reset control

**PDL\_WDT\_RESET\_DISABLE** or  
PDL\_WDT\_RESET\_ENABLE

Disable or enable reset of the MCU when the watchdog timer overflows with no callback function specified.

**[func]**

The function to be called at the periodic interval.

Specify PDL\_NO\_FUNC to have the timer output a WDTOVF# signal. The MCU will also be reset (if selected above).

**[data2]**

The interrupt priority level. Select between 1 (lowest priority) and 15 (highest priority). This parameter will be ignored if PDL\_NO\_FUNC is specified for parameter func.

**Return value**

True if all parameters are valid and exclusive; otherwise false.

**Category**

Watchdog Timer

**Reference**

R\_CGC\_Set

**Remarks**

- Function R\_CGC\_Set should be called before any use of this function.
- If a callback function is specified, this function will enable the relevant interrupt. Please see the notes on callback function use in §6.
- A callback function is executed by the interrupt processing function. This means that no other interrupt can be processed until the callback function has completed.
- The timing limits depend on the frequency of the peripheral module clock, PCLK.

$$Period = \frac{n \times 256}{f_{PCLK}} \text{ or } Frequency = \frac{f_{PCLK}}{n \times 256}$$

Where n = 4, 64, 128, 512, 2048, 8192, 32768 or 131072.  
Examples for different values of f<sub>PCLK</sub> are given below.

	f <sub>PCLK</sub> (MHz)			
	50	12.5	32	8
Period <sub>PCLK÷4</sub>	20.5 μs	81.9 μs	32.0 μs	128 μs
Period <sub>PCLK÷64</sub>	328 μs	1.31 ms	512.0 μs	2.05 ms
Period <sub>PCLK÷128</sub>	655 μs	2.62 ms	1.02 ms	4.10 ms
Period <sub>PCLK÷512</sub>	2.62 ms	10.5 ms	4.10 ms	16.4 ms
Period <sub>PCLK÷2048</sub>	10.5 ms	41.9 ms	16.4 ms	65.5 ms
Period <sub>PCLK÷8192</sub>	41.9 ms	168 ms	65.5 ms	262 ms
Period <sub>PCLK÷32768</sub>	168 ms	671 ms	262 ms	1.05 s
Period <sub>PCLK÷131072</sub>	671 ms	2.68 s	1.05 s	4.19 s
f <sub>PCLK÷4</sub>	48.8 kHz	12.2 kHz	31.3 kHz	7.81 kHz
f <sub>PCLK÷64</sub>	3.05 kHz	763 Hz	1.95 kHz	488 Hz
f <sub>PCLK÷128</sub>	1.53 kHz	381 Hz	977 Hz	244 Hz
f <sub>PCLK÷512</sub>	381 Hz	95.4 Hz	244 Hz	61.0 Hz
f <sub>PCLK÷2048</sub>	95.4 Hz	23.8 Hz	61.0 Hz	15.3 Hz
f <sub>PCLK÷8192</sub>	23.8 Hz	5.96 Hz	15.3 Hz	3.81 Hz
f <sub>PCLK÷32768</sub>	5.96 Hz	1.49 Hz	3.81 Hz	0.954 Hz
f <sub>PCLK÷131072</sub>	1.49 Hz	0.373 Hz	0.954 Hz	0.238 Hz

**Program example**

```

/* RPDL definitions */
#include "r_pdl_wdt.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Configure the watchdog timer for PCLK/4 operation */
    R_WDT_Create(
        PDL_WDT_PCLK_DIV_4,
        WDT_handler,
        7
    );

    /* Configure the watchdog timer for PCLK/131072 operation with output
    and reset enable */
    R_WDT_Create(
        PDL_WDT_PCLK_DIV_131072 | PDL_WDT_RESET_ENABLE,
        PDL_NO_FUNC,
        0
    );
}

```



## 2) R\_WDT\_Control

### Synopsis

Control the Watchdog operation.

### Prototype

```
bool R_WDT_Control(
    uint8_t data // Control selection
);
```

### Description

Modify the operation of the Watchdog timer.

#### [data]

Configure the timer channel. To set multiple options at the same time, use "|" to separate each value.

- Counter stop

PDL_WDT_STOP	Disable the counter clock source.
--------------	-----------------------------------

- Counter update

PDL_WDT_RESET_COUNTER	Reset the counter.
-----------------------	--------------------

### Return value

True if all parameters are valid and exclusive; otherwise false.

### Category

Watchdog Timer

### Reference

R\_WDT\_Create

### Remarks

- R\_WDT\_Create must be first be used to configure the timer.

### Program example

```
/* RPDL definitions */
#include "r_pdl_wdt.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Prevent the watchdog timer from overflowing */
    R_WDT_Control(
        PDL_WDT_RESET_COUNTER
    );
}
```

**3) R\_WDT\_Read****Synopsis**

Read the Watchdog timer status register.

**Prototype**

```
bool R_WDT_Read(
    uint8_t * data,    // A pointer to the data storage location
);
```

**Description**

Read and store the status flag.

**[data]**

The timer status shall be stored in the following format.

b7 – b1	b0
0	0: Not overflowed 1: An overflow has occurred

**Return value**

True if all parameters are valid; otherwise false.

**Category**

Watchdog Timer

**Reference**

R\_WDT\_Create

**Remarks**

- If the flag is set to 1, it shall be automatically cleared to 0 by this function.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_wdt.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

uint8_t Flags;

void func(void)
{
    /* Read the timer values */
    R_WDT_Read(
        &Flags
    );
}
```

## 4.2.15. Independent Watchdog Timer

## 1) R\_IWDT\_Set

**Synopsis**

Configure the Independent Watchdog operation.

**Prototype**

```
bool R_IWDT_Set(
    uint16_t data    // Configuration selection
);
```

**Description**

Select the operation of the Independent Watchdog timer.

**[data]**

Configure the timer options. Use “|” to separate each value.

- Counter selection

PDL_IWDT_TIMEOUT_1024 or PDL_IWDT_TIMEOUT_4096 or PDL_IWDT_TIMEOUT_8192 or PDL_IWDT_TIMEOUT_16384	The number of cycles of the selected clock before the reset occurs.
PDL_IWDT_CLOCK_OCO_1 or PDL_IWDT_CLOCK_OCO_16 or PDL_IWDT_CLOCK_OCO_32 or PDL_IWDT_CLOCK_OCO_64 or PDL_IWDT_CLOCK_OCO_128 or PDL_IWDT_CLOCK_OCO_256	The selected clock frequency. The low-speed on-chip oscillator clock (IWDTCLK) ÷ 1, 16, 32, 64, 128 or 256.

**Return value**

True if all parameters are valid and exclusive; otherwise false.

**Category**

Independent Watchdog Timer

**Reference**

R\_IWDT\_Control

**Remarks**

- R\_IWDT\_Control must be used to start the timer.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_iwdt.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Configure the IWDT */
    R_IWDT_Set(
        PDL_IWDT_TIMEOUT_16384 | PDL_IWDT_CLOCK_OCO_256
    );
}
```

## 2) R\_IWDT\_Control

<b>Synopsis</b>	Control the Independent Watchdog operation.		
<b>Prototype</b>	<pre>bool R_IWDT_Control(     uint8_t data    // Control selection );</pre>		
<b>Description</b>	<p>Modify the operation of the Independent Watchdog timer.</p> <p><b>[data]</b> Control the timer.</p> <ul style="list-style-type: none"> <li>Counter start / refresh</li> </ul> <table border="1"> <tr> <td>PDL_IWDT_REFRESH</td><td>Start or refresh the counter by re-loading the timeout value.</td></tr> </table>	PDL_IWDT_REFRESH	Start or refresh the counter by re-loading the timeout value.
PDL_IWDT_REFRESH	Start or refresh the counter by re-loading the timeout value.		
<b>Return value</b>	True if the parameter is valid; otherwise false.		
<b>Category</b>	Independent Watchdog Timer		
<b>Reference</b>	R_IWDT_Set		
<b>Remarks</b>	<ul style="list-style-type: none"> <li>R_IWDT_Set must be first be used to configure the timer.</li> </ul>		
<b>Program example</b>	<pre>/* RPDL definitions */ #include "r_pdl_iwdt.h"  /* RPDL device-specific definitions */ #include "r_pdl_definitions.h"  void func(void) {     /* Refresh the IWDT */     R_IWDT_Control(         PDL_IWDT_REFRESH     ); }</pre>		

### 3) R\_IWDT\_Read

**Synopsis**

Read the watchdog timer status and counter.

**Prototype**

```
bool R_IWDT_Read(
    uint16_t * data // A pointer to the data storage location
);
```

**Description**

Read and store the status flags.

**[data]**

The timer status shall be stored in the following format.

b15	b14	b13 - b0
0	0: Not underflowed 1: An underflow has occurred	The current counter value

**Return value**

True.

**Category**

Independent Watchdog Timer

**Reference**

None.

**Remarks**

- If the flag is set to 1, it shall be automatically cleared to 0 by this function.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_iwdt.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

uint16_t Status;

void func(void)
{
    /* Read the timer status */
    R_IWDT_Read(
        &Status
    );
}
```

## 4.2.16. Serial Communication Interface

## 1) R\_SCI\_Set

**Synopsis**

Configure the SCI pin selection.

**Prototype**

```
bool R_SCI_Set(
    uint8_t data // Configuration
);
```

**Description**

Set up the global SCI options.

**[data]**

Configure the global options.

- Pin selection (required only if the pins are used for the SCI function).

PDL_SCI_PIN_SCI2_A or PDL_SCI_PIN_SCI2_B	Select the -A or -B pins for Rx/D2, SCK2, Tx/D2.
---	--

**Return value**

True if all parameters are valid and exclusive; otherwise false.

**Category**

SCI

**Reference**

R\_SCI\_Create

**Remarks**

- Before calling R\_SCI\_Create, if the selected device package offers A or B pins for SCI signals call this function to configure the relevant pins.
- Pins which are not used for the SCI functions may be omitted.
- Please refer to the "Port Function Control Register F (PFFSCI)" section in the RX62G Hardware Manual for details of SCI pin selection.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_sci.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Configure the applicable SCI pins */
    R_SCI_Set(
        PDL_SCI_PIN_SCI2_A
    );
}
```

## 2) R\_SCI\_Create

### Synopsis

SCI channel setup.

### Prototype

```
bool R_SCI_Create(
    uint8_t data1,    // Channel selection
    uint32_t data2,    // Channel configuration
    uint32_t data3,    // Bit rate or register value
    uint8_t data4      // Interrupt priority level
);
```

### Description (1/3)

Set up the selected SCI channel.

#### [data1]

Select channel SCIn (where n = 0 to 2).

#### [data2]

Configure the channel. If multiple selections are required, use “|” to separate each selection. The default settings are shown in **bold**.

- Operation mode

PDL_SCI_ASYNC or PDL_SCI_SYNC or PDL_SCI_SMART or PDL_SCI_ASYNC_MP	Choose between Asynchronous, Clock synchronous, Smart Card Interface or Multi-Processor Asynchronous operation.
---	--

- Transmit / Receive connections

<b>PDL_SCI_TX_CONNECTED</b> or PDL_SCI_TX_DISCONNECTED or <b>PDL_SCI_RX_CONNECTED</b> or PDL_SCI_RX_DISCONNECTED	The TXDn output is required / not required.
	The RXDn input is required / not required.

- Data transfer format

<b>PDL_SCI_LSB_FIRST</b> or PDL_SCI_MSB_FIRST	Select least- or most-significant bit first.
--	--

#### Options which are available in Asynchronous mode

- Data clock source selection

<b>PDL_SCI_CLK_INT_IO</b> or PDL_SCI_CLK_INT_OUT or PDL_SCI_CLK_EXT	Select the on-chip baud rate generator.	The SCKn pin functions as an I/O pin. The SCKn pin outputs the bit clock.
	Input a clock of 8 or 16 times the desired bit rate to the SCKn pin. See parameter data3 for the multiplier selection.	

- Noise cancellation

<b>PDL_SCI_NC_ENABLE</b> or PDL_SCI_NC_DISABLE	Control noise cancellation.
---	-----------------------------

- Data length

<b>PDL_SCI_8_BIT_LENGTH</b> or PDL_SCI_7_BIT_LENGTH	8- or 7-bit data length.
--	--------------------------

- Parity mode

<b>PDL_SCI_PARITY_NONE</b> or PDL_SCI_PARITY_EVEN or PDL_SCI_PARITY_ODD	No parity bit, even parity bit or odd parity bit. Note: Do not select the parity bit for Multi-Processor Asynchronous mode.
---	---

- Stop bit length

<b>PDL_SCI_STOP_1</b> or PDL_SCI_STOP_2	One or two stop bits.
--	-----------------------

The option “PDL\_SCI\_8N1” can be used to select 8-bit data length, no parity and one stop bit.

**Description (2/3)**Options which are available in Clock Synchronous mode

- Data clock source selection

<b>PDL_SCI_CLK_INT_OUT</b> or	Select the On-chip baud rate generator. The SCKn pin outputs the bit clock.
<b>PDL_SCI_CLK_EXT</b>	Input the clock to the SCKn pin.

Options which are available in Smart Card Interface mode

- Data inversion

<b>PDL_SCI_INVERSION_OFF</b> or <b>PDL_SCI_INVERSION_ON</b>	Control data inversion (transmission and reception).
--	--

- Base clock pulse cycle count

<b>PDL_SCI_BCP_32</b> or <b>PDL_SCI_BCP_64</b> or <b>PDL_SCI_BCP_93</b> or <b>PDL_SCI_BCP_128</b> or <b>PDL_SCI_BCP_186</b> or <b>PDL_SCI_BCP_256</b> or <b>PDL_SCI_BCP_372</b> or <b>PDL_SCI_BCP_512</b>	The number of base clock cycles in a 1-bit data transfer period.
--	--

- Parity selection

<b>PDL_SCI_PARITY_EVEN</b> or <b>PDL_SCI_PARITY_ODD</b>	Select even or odd parity bit.
--	--------------------------------

- Block transfer mode selection

<b>PDL_SCI_BLOCK_MODE_OFF</b> or <b>PDL_SCI_BLOCK_MODE_ON</b>	Control Block transfer mode.
--	------------------------------

- GSM mode selection

<b>PDL_SCI_GSM_MODE_OFF</b> or <b>PDL_SCI_GSM_MODE_ON</b>	Control GSM mode.
--	-------------------

- SCKn pin output control

	Normal mode	GSM mode
<b>PDL_SCI_SCK_OUTPUT_OFF</b> or	I/O pin	Not applicable
<b>PDL_SCI_SCK_OUTPUT_LOW</b> or	Not applicable.	Fixed low.
<b>PDL_SCI_SCK_OUTPUT_ON</b> or	Outputs the bit clock.	
<b>PDL_SCI_SCK_OUTPUT_HIGH</b>	Not applicable	Fixed high.

**[data3]**

Select the SCI transfer rate.

See the Remarks section for the maximum rate that the device can support.

The format may be either:

- The transfer bit rate in bits per second (bps).  
The clock division values will be calculated using this value.  
This format is valid only when the on-chip baud rate generator is selected as the data clock source (in parameter data2).

Or the following, using “|” to separate each selection.

- b31    b30 – b24    |    b23 – b0  

1	0	A value between 256 (0x100) and 16,776,960 (0xFFFF00) that is nearest to the transfer bit rate.
---	---	---

- ABCS selection (required for asynchronous mode)

<b>PDL_SCI_CYCLE_BIT_16</b> or <b>PDL_SCI_CYCLE_BIT_8</b>	Select 16 or 8 base clock cycles for one bit period.
--	--



**Description (3/3)**

- CKS selection (required if the on-chip baud rate generator is selected as the data clock source)

PDL\_SCI\_PCLK\_DIV\_1 or  
PDL\_SCI\_PCLK\_DIV\_4 or  
PDL\_SCI\_PCLK\_DIV\_16 or  
PDL\_SCI\_PCLK\_DIV\_64

Select the internal clock signal PCLK ÷ 1, 4, 16 or 64 as the baud rate generator clock source.

- BRR setting (required if the on-chip baud rate generator is selected as the data clock source)  
The BRR register value, between 0 and 255.

**[data4]**

The interrupt priority level. Select between 1 (lowest priority) and 15 (highest priority). This parameter will be ignored if PDL\_NO\_FUNC is specified for parameter func in functions R\_SCI\_Send or R\_SCI\_Receive.

**Return value**

True if all parameters are valid, exclusive and achievable; otherwise false.

**Category**

SCI

**Reference**

R\_SCI\_Destroy, R\_SCI\_Set, R\_SCI\_Send, R\_SCI\_Receive

**Remarks**

- Function R\_CGC\_Set must be called before any use of this function.
- This function configures each SCI pin that is required for operation. It also disables the alternative modes on those pins.
- The wait time of 1 data bit period that is required during configuration is handled within this function.
- The range of achievable bit rates is listed below.

Mode	Data clock source	Limit	f <sub>PCLK</sub>			
			50 MHz	12.5 MHz	32 MHz	8 MHz
Asynchronous	Internal	Minimum	96	24	62	16
	External	Maximum	3,125,000	781,250	2,000,000	500,000
Synchronous	Internal	Minimum	763	191	489	123
	External	Maximum	6,250,000	1,562,500	4,000,000	1,000,000
Smart card	Internal	Minimum	3	1	2	1
		Maximum	781,250	195,312	500,000	125,000

**Program example**

```

/* RPDL definitions */
#include "r_pdl_sci.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Configure SCI0 for asynchronous, 8N1, 38400 baud */
    R_SCI_Create(
        0,
        PDL_SCI_ASYNC | PDL_SCI_8N1,
        38400,
        1
    );

    /* Configure SCI1 for asynchronous, 8N1, register values supplied */
    R_SCI_Create(
        1,
        PDL_SCI_ASYNC | PDL_SCI_8N1,
        BIT_31 | PDL_SCI_PCLK_DIV_1 | PDL_SCI_CYCLE_BIT_16 | \
        (115200 & 0x00FFFF00) | 0x50,
        1
    );
}

```

### 3) R\_SCI\_Destroy

**Synopsis**

Shut down a SCI channel.

**Prototype**

```
bool R_SCI_Destroy(
    uint8_t data    // Channel selection
);
```

**Description**

Stop data flow and shutdown the selected SCI channel.

**[data]**

Select channel SCIn (where n = 0 to 2).

**Return value**

True if all parameters are valid; otherwise false.

**Category**

SCI

**Reference**

R\_SCI\_Create

**Remarks**

- The SCI channel is put into the power-down state.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_sci.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Shutdown SCI channel 1 */
    R_SCI_Destroy(
        1
    );
}
```

## 4) R\_SCI\_Send

**Synopsis**

Transmit data on a SCI channel.

**Prototype**

```

bool R_SCI_Send(
    uint8_t data1,    // Channel selection
    uint8_t data2,    // Channel configuration (and Target Station ID)
    uint8_t * data3,  // Data start address
    uint16_t data4,   // Data count
    void * func       // Callback function
);

```

**Description**

Transmit data on the specified serial channel.

**[data1]**

Select channel SCIn (where n = 0 to 2).

**[data2]**

Control options.

The default options are shown in **bold**. Specify PDL\_NO\_DATA to use the defaults.

- DTC trigger control.

<b>PDL_SCI_DTC_TRIGGER_DISABLE</b> or <b>PDL_SCI_DTC_TRIGGER_ENABLE</b>	Disable or enable activation of the DTC when a data byte is transmitted.
--	--

- ID transmission control (valid only in Multi-processor mode).

<b>PDL_SCI_MP_ID_CYCLE</b>	Transmit the upper byte as the ID byte. The valid ID range is 0 to 255.
----------------------------	--

**[data3]**

The start address of the data to be sent.

Specify PDL\_NO\_PTR for the ID cycle in Multi-processor mode.

If the DTC shall be used to transfer the data, specify PDL\_NO\_PTR.

**[data4]**

For sending binary data, set this to the number of bytes to be sent. The valid range is 1 to 65535.

Set this to 0 for transmission of a null-terminated character string.

For the ID cycle in Multi-processor mode, specify 0.

If the DTC shall be used to transfer the data, specify PDL\_NO\_DATA.

**[func]**

Specify PDL\_NO\_FUNC or a callback function name, depending on the required transfer method.

Use R\_SCI\_Control to terminate this operation early.

R\_SCI\_GetStatus can be used to find out how many characters have been transmitted.

Transfer method	Parameter
Polling	PDL_NO_FUNC. This function will continue until the required number of bytes has been sent.
Interrupts	The function to be called when the last byte has been sent.
DTC	The function to be called at the interval specified in R_DTC_Create.

**Return value**

True if all parameters are valid and the operation completed without errors;  
False if a parameter was out of range or if the channel was already transmitting or if an error occurred during transmission.

**Category**

SCI

**Reference**

R\_SCI\_Create, R\_SCI\_Control, R\_SCI\_GetStatus

**Remarks**

- The compiler adds a null character to the end of string constants.
  - If a callback function is specified, transmission interrupts are used. Please see the notes on callback function usage in §6.
  - If polling mode is used, the TXI and TEND flags will be used to manage the data transmission. If the SCI channel's control registers are directly modified by the user, this function may lock up.
  - The maximum number of characters to be transmitted is 65535.
  - A callback function is executed by the interrupt processing function. This means that no other interrupt can be processed until the callback function has completed.
  - If reception is enabled and receive errors occur, transmission will be blocked until the errors are cleared.
  - In Multi-processor mode, R\_SCI\_Send is to be called in pair: the first one is to send ID (ID cycle); the second one is to send data (Data cycle). For ID transmission, it will be sent by internal polling operation. For data transmission, it will be the same as normal Asynchronous mode.
- For a usage example of Multi-processor mode, please refer to Section 5.9.5.
- For ID cycle, the DTC trigger control and the callback function will be ignored.

**Program example**

```

/* RPDL definitions */
#include "r_pdl_sci.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func( void )
{
    uint8_t data_store[100];

    /* Send a string on channel 2 */
    R_SCI_Send(
        2,
        PDL_NO_DATA,
        "Renesas RX",
        0,
        PDL_NO_FUNC
    );

    /* Send 50 bytes of binary data on channel 1 */
    R_SCI_Send(
        2,
        PDL_NO_DATA,
        data_store,
        50,
        PDL_NO_FUNC
    );

    /* Send the ID byte (0x0A, shifted into the upper byte) */
    R_SCI_Send(
        2,
        PDL_SCI_MP_ID_CYCLE | 0x0A00,
        PDL_NO_PTR,
        0,
        PDL_NO_FUNC
    );
}

```

## 5) R\_SCI\_Receive

**Synopsis**

Receive data on a SCI channel.

**Prototype**

```

bool R_SCI_Receive(
    uint8_t data1,    // Channel selection
    uint8_t data2,    // Channel configuration (and Station ID of receiving device)
    uint8_t * data3,  // Data start address
    uint16_t data4,   // Receive threshold
    void * func1,     // Callback function
    void * func2      // Callback function
);

```

**Description**

Enable SCI reception and acquire any incoming data.

**[data1]**

Select channel SCIn (where n = 0 to 2).

**[data2]**

Control options.

The default options are shown in **bold**. Specify PDL\_NO\_DATA to use the defaults.

- DTC trigger control

The default setting is shown in **bold**. May also specify PDL\_NO\_DATA to use the defaults.

<b>PDL_SCI_DTC_TRIGGER_DISABLE</b> or PDL_SCI_DTC_TRIGGER_ENABLE	Disable or enable activation of the DTC when a data byte is received.
---	---

- ID reception control (valid only in Multi-processor mode).

PDL_SCI_MP_ID_CYCLE	Use the upper byte as the station ID. The valid ID range is 0 to 255.
---------------------	--

**[data3]**

The start address of the storage area for the expected data.

Specify PDL\_NO\_PTR if no data shall be processed by this function e.g. if the DTC shall be used to process the received data, or for ID cycle in Multi-processor mode.

**[data4]**

The number of bytes that must be received before the function completes or the callback function is called.

Specify 0 for the ID cycle in Multi-processor mode.

If the DTC shall be used to handle the received data, specify PDL\_NO\_DATA.

**[func1]**

Specify PDL\_NO\_FUNC or a callback function name, depending on the required transfer method.

Transfer method	Parameter
Polling	PDL_NO_FUNC. This function will continue until the required number of bytes has been received.
Interrupts	The function to be called when the number of received bytes reaches the threshold number.
DTC	The function to be called at the interval specified in R_DTC_Create.

**[func2]**

The function to be called if a receive error occurs. Specify PDL\_NO\_FUNC to ignore errors.

**Return value**

True if all parameters are valid and the operation completed; false if a parameter was out of range.

**Category**

SCI

**Reference**

R\_SCI\_Create, R\_SCI\_Control, R\_SCI\_GetStatus

**Remarks**

- The maximum number of characters to be received is 65535.
- Wait until a transmission on the same channel is complete before calling this function.
- If callback function func1 is specified, reception interrupts are used.  
Please see the notes on callback function usage in §6.
- If polling mode is used, the RXI flag will be used to manage the data reception.  
If the SCI channel's control registers are directly modified by the user, this function may lock up.
- If no error callback function func2 is specified, the error flags are cleared automatically to allow the reception process to complete.
- Callback functions are executed by the interrupt processing function. This means that no other interrupt can be processed until a callback function has completed.
- In Multi-processor mode, R\_SCI\_Receive is to be called in a pair: the first one is to receive ID (ID cycle); the second one is to receive data (Data cycle). For ID reception, it could be done by reception interrupt (by specifying func1), or by internal polling operation (without specifying func1). For data reception, it will be the same as normal Asynchronous mode.  
For a usage example of Multi-processor mode, please refer to Section 5.9.4.
- For the ID cycle, the DTC trigger control will be ignored.
- If synchronous reception and transmission are required, a transmission must be started at the same time as reception. Please refer to the usage example in section 5.9.3.

**Program example**

```

/* RPDL definitions */
#include "r_pdl_sci.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

volatile uint8_t SCIIReceiveBuffer[10];

/* SCI channel 1 receive data handler */
void SCII RxFunc(void){}

/* SCI channel 1 error handler */
void SCII ErrFunc(void){}

void func( void )
{
    uint8_t temp;

    /* Wait for 1 character to be received on channel 0 */
    R_SCI_Receive(
        0,
        PDL_NO_DATA,
        &temp,
        1,
        PDL_NO_FUNC,
        PDL_NO_FUNC
    );

    /* Start the reception of 9 characters on channel 1 */
    R_SCI_Receive(
        1,
        PDL_NO_DATA,
        SCIIReceiveBuffer,
        9,
        SCII RxFunc,
        SCII ErrFunc
    );
}

```

## 6) R\_SCI\_Control

### Synopsis

Control the SCI channel.

### Prototype

```
bool R_SCI_Control(
    uint8_t data1, // Channel selection
    uint8_t data2  // Channel control
);
```

### Description

Stops SCI transmission or reception.

#### [data1]

Select channel SCIn (where n = 0 to 2).

#### [data2]

Control the channel. If multiple selections are required, use "|" to separate each selection.

- Select the process to be stopped.

PDL_SCI_STOP_TX	Stop the transmission process. If a reception process is active, the transmit output will not become idle until the reception process has stopped.
PDL_SCI_STOP_RX	Stop the reception process. If a transmission process is active, the receive error flags may be set erroneously. These can be ignored and will be cleared when a new reception process is started.

The option "PDL\_SCI\_STOP\_TX\_AND\_RX" can be used to select both processes.

If both processes are selected, transmission and reception will stop immediately.

- Generate a Space or Mark signal when idle.

PDL_SCI_OUTPUT_SPACE	Set the idle output to Space (logic 0). This can be used to generate a Break condition.
PDL_SCI_OUTPUT_MARK	Set the idle output to Mark (logic 1).

- Error flag control

PDL_SCI_CLEAR_RECEIVE_ERROR_FLAGS	Try to clear the receive error flags.
-----------------------------------	---------------------------------------

- Manual SCK control

PDL_SCI_GSM_SCK_STOP or PDL_SCI_GSM_SCK_START	Disable or enable the clock output (can be used while GSM mode is enabled).
--	---

### Return value

True if all parameters are valid; otherwise false.

### Category

SCI

### Reference

R\_SCI\_Send, R\_SCI\_Receive, R\_SCI\_GetStatus

### Remarks

- None.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_sci.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Terminate SCI reception on channel 0 */
    R_SCI_Control(
        0,
        PDL_SCI_STOP_RX
    );
}
```



## 7) R\_SCI\_GetStatus

**Synopsis**

Check the status of a SCI channel.

**Prototype**

```
bool R_SCI_GetStatus(
    uint8_t data1,      // Channel selection
    uint8_t * data2,    // Status flags
    uint8_t * data3,    // Last byte received
    uint16_t * data4,   // Bytes transmitted
    uint16_t * data5    // Bytes received
);
```

**Description**

Acquires the channel status and the byte counts

**[data1]**

Select channel SCIn (where n = 0 to 2).

**[data2]**

The status flags shall be stored in the format:  
Asynchronous or Synchronous mode:

b7	b6	b5	b4	b3	b2	b1	b0
Buffer status		Reception error detection			Transmit status	0	RxD pin level
Transmit	Receive	Overrun	Framing	Parity			
0: Full 1: Empty	0: Empty 1: Full		0: No error 1: Detected		0: Active 1: Idle		0: Low 1: High

Smart card mode:

b7	b6	b5	b4	b3	b2	b1	b0
Buffer status		Error detection			Transmit status	0	RxD pin level
Transmit	Receive	Overrun	Error signal	Parity			
0: Full 1: Empty	0: Empty 1: Full		0: No error 1: Detected		0: Active 1: Idle		0: Low 1: High

**[data3]**

The storage location for the last byte that was received. Specify PDL\_NO\_PTR if this information is not required.

**[data4]**

The storage location for the number of characters that are have been transmitted in the current transmission. Specify PDL\_NO\_PTR if this information is not required.

**[data5]**

The storage location for the number of characters that are have been received in the current reception process. Specify PDL\_NO\_PTR if this information is not required.

**Return value**

True if all parameters are valid and the operation completed; false if a parameter was out of range.

**Category**

SCI

**Reference**

R\_SCI\_Send, R\_SCI\_Receive

**Remarks**

- The error flags are not modified by this function. They are cleared when a new reception process is started.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_sci.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

uint8_t StatusValue;
uint16_t TxChars;
uint16_t RxChars;

void func(void)
{
    /* Read the status of SCI channel 0 */
    R_SCI_GetStatus(
        0,
        &StatusValue,
        PDL_NO_PTR,
        &TxChars,
        &RxChars
    );
}
```

## 4.2.17. CRC calculator

## 1) R\_CRC\_Create

**Synopsis**

Configure the CRC calculator.

**Prototype**

```
bool R_CRC_Create(
    uint8_t data    // Configuration
);
```

**Description**

Enable the CRC and set the operating conditions.

**[data]**

Calculation options. To set multiple options at the same time, use "|" to separate each value.

- Polynomial selection

PDL_CRC_POLY_CRC_8 or	$X^8 + X^2 + X + 1$
PDL_CRC_POLY_CRC_16 or	$X^{16} + X^{15} + X^2 + 1$
PDL_CRC_POLY_CRC_CCITT	$X^{16} + X^{12} + X^5 + 1$

- Bit order

PDL_CRC_LSB_FIRST or PDL_CRC_MSB_FIRST	Select LSB or MSB-first operation.
---	------------------------------------

**Return value**

True if all parameters are valid and exclusive; otherwise false.

**Category**

CRC

**References****Remarks**

- None.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_crc.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Set up the CRC in 8-bit mode with LSB first */
    R_CRC_Create(
        PDL_CRC_POLY_CRC_8 | PDL_CRC_LSB_FIRST
    );
}
```

## 2) R\_CRC\_Destroy

**Synopsis**

Shut down the CRC calculator.

**Prototype**

```
bool R_CRC_Destroy(  
    void           // No parameter is required  
);
```

**Description**

Put the CRC calculator into the Power-down state, with minimal power consumption.

**Return value**

True.

**Category**

CRC

**Reference**

R\_CRC\_Create

**Remarks**

- None.

**Program example**

```
/* RPDL definitions */  
#include "r_pdl_crc.h"  
  
/* RPDL device-specific definitions */  
#include "r_pdl_definitions.h"  
  
void func( void )  
{  
    /* Shut down the CRC */  
    R_CRC_Destroy(  
    );  
}
```

### 3) R\_CRC\_Write

**Synopsis**

Write data into the CRC calculation register.

**Prototype**

```
bool R_CRC_Write(  
    uint8_t data    // The data to be used for the calculation  
);
```

**Description**

Write the data into the data input register.

**[data]**

The data to be written into the register.

**Return value**

True.

**Category**

CRC

**Reference**

R\_CRC\_Create

**Remarks**

- None.

**Program example**

```
/* RPDL definitions */  
#include "r_pdl_crc.h"  
  
/* RPDL device-specific definitions */  
#include "r_pdl_definitions.h"  
  
void func(void)  
{  
    /* Write F0h into the CRC calculation register */  
    R_CRC_Write(  
        0xF0  
    );  
}
```

## 4) R\_CRC\_Read

**Synopsis**

Read the CRC calculation result.

**Prototype**

```
bool R_CRC_Read(
    uint8_t data1,    // Control
    uint16_t * data2  // Data storage location
);
```

**Description**

Reads and stores the CRC calculation result.

**[data1]**

Control the behaviour of the CRC unit.

The default setting is shown in **bold**. Specify PDL\_NO\_DATA to use the default.

- Result register clearing

<b>PDL_CRC_CLEAR_RESULT</b> or PDL_CRC_RETAIN_RESULT	Clear or retain the value in the result register.
---	---

**[data2]**

The address of the location where the result shall be stored.

For the 8-bit polynomial, the results are stored in the lower-order byte.

**Return value**

True.

**Category**

CRC

**Reference**

None.

**Remarks**

- None.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_crc.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    uint16_t CRCresult;

    /* Read the CRC result and clear it */
    R_CRC_Read(
        PDL_CRC_RETAIN_RESULT,
        &CRCresult
    );
}
```

4.2.18. I<sup>2</sup>C Bus Interface

## 1) R\_IIC\_Create

**Synopsis**I<sup>2</sup>C channel setup.**Prototype**

```
bool R_IIC_Create(
    uint8_t data1, // Channel selection
    uint32_t data2, // Channel configuration
    uint32_t data3, // Detection configuration
    uint16_t data4, // Slave address
    uint16_t data5, // Slave address
    uint16_t data6, // Slave address
    uint32_t data7, // Transfer rate control
    uint32_t data8 // Rise and fall time correction
);
```

**Description (1/3)**Set up the selected I<sup>2</sup>C channel.**[data1]**

Select channel IICn (where n = 0).

**[data2]**Configure the channel. If multiple selections are required, use “|” to separate each selection. The default settings are shown in **bold**.

- Bus mode selection

PDL_IIC_MODE_IIC or PDL_IIC_MODE_SMBUS	Choose I <sup>2</sup> C Bus or SMBus mode.
---	---

- Internal reference clock

<b>PDL_IIC_INT_PCLK_DIV_1</b> or PDL_IIC_INT_PCLK_DIV_2 or PDL_IIC_INT_PCLK_DIV_4 or PDL_IIC_INT_PCLK_DIV_8 or PDL_IIC_INT_PCLK_DIV_16 or PDL_IIC_INT_PCLK_DIV_32 or PDL_IIC_INT_PCLK_DIV_64 or PDL_IIC_INT_PCLK_DIV_128	The reference clock source, used inside the I <sup>2</sup> C module.
---	--

- Timeout detection control

<b>PDL_IIC_TIMEOUT_DISABLE</b> or PDL_IIC_TIMEOUT_LOW or PDL_IIC_TIMEOUT_HIGH or PDL_IIC_TIMEOUT_BOTH	Disable timeout detection, or enable for SCL stuck at a low level high level or both low and high level.
--	---

- Timeout mode

<b>PDL_IIC_TIMEOUT_LONG</b> or PDL_IIC_TIMEOUT_SHORT	Select 16-bit (long) or 14-bit (short) mode.
---	---

- SDA output delay count

<b>PDL_IIC_SDA_DELAY_0</b> or PDL_IIC_SDA_DELAY_1 or PDL_IIC_SDA_DELAY_2 or PDL_IIC_SDA_DELAY_3 or PDL_IIC_SDA_DELAY_4 or PDL_IIC_SDA_DELAY_5 or PDL_IIC_SDA_DELAY_6 or PDL_IIC_SDA_DELAY_7	Select the number of cycles for the SDA output delay counter.
--	--

- SDA output delay clock source

<b>PDL_IIC_SDA_DELAY_DIV_1</b> or PDL_IIC_SDA_DELAY_DIV_2	Select the clock source (internal reference clock ÷ 1 or 2) for the SDA output delay counter.
--	--

**Description (2/3)**

- Noise filter control

**PDL\_IIC\_NF\_1** or  
**PDL\_IIC\_NF\_2** or  
**PDL\_IIC\_NF\_3** or  
**PDL\_IIC\_NF\_4** or  
**PDL\_IIC\_NF\_DISABLE**

Select the number of stages in the noise filter.

**[data3]**

Detection settings. Specify PDL\_NO\_DATA to use the defaults.

- NACK Transmission Arbitration Lost Detection control

**PDL\_IIC\_NTALD\_DISABLE** or  
**PDL\_IIC\_NTALD\_ENABLE**

Disable or enable arbitration to be lost when an ACK is detection during transmission of a NACK in receive mode.

- Slave Arbitration Lost Detection control

**PDL\_IIC\_SALD\_DISABLE** or  
**PDL\_IIC\_SALD\_ENABLE**

Disable or enable arbitration to be lost when a mismatch occurs during slave data transmission.

- Slave address detection control

**PDL\_IIC\_SLAVE\_0\_DISABLE** or  
**PDL\_IIC\_SLAVE\_0\_ENABLE\_7** or  
**PDL\_IIC\_SLAVE\_0\_ENABLE\_10**

Disable or enable detection of slave address 0 in 7-bit or 10-bit format.

**PDL\_IIC\_SLAVE\_1\_DISABLE** or  
**PDL\_IIC\_SLAVE\_1\_ENABLE\_7** or  
**PDL\_IIC\_SLAVE\_1\_ENABLE\_10**

Disable or enable detection of slave address 1 in 7-bit or 10-bit format.

**PDL\_IIC\_SLAVE\_2\_DISABLE** or  
**PDL\_IIC\_SLAVE\_2\_ENABLE\_7** or  
**PDL\_IIC\_SLAVE\_2\_ENABLE\_10**

Disable or enable detection of slave address 2 in 7-bit or 10-bit format.

**PDL\_IIC\_SLAVE\_GCA\_DISABLE** or  
**PDL\_IIC\_SLAVE\_GCA\_ENABLE**

Disable or enable detection of the General Call address.

- Device-ID detection control

**PDL\_IIC\_DEVICE\_ID\_DISABLE** or  
**PDL\_IIC\_DEVICE\_ID\_ENABLE**

Disable or enable detection of the Device-ID address (1111 100b).

- Host Address detection control

**PDL\_IIC\_HOST\_ADDRESS\_DISABLE** or  
**PDL\_IIC\_HOST\_ADDRESS\_ENABLE**

Disable or enable detection of the SMBus host address.

**[data4]**

Slave address 0. Ignored if slave address 0 detection is disabled.

**[data5]**

Slave address 1. Ignored if slave address 1 detection is disabled.

**[data6]**

Slave address 2. Ignored if slave address 2 detection is disabled.

**[data7]**

Transfer rate control.

Either:

The maximum bit rate in bits per second.

For Master mode, the clock division values will be calculated using a 50% duty cycle.

For Slave mode, the rate will be used to calculate the clock stretching period.

Or:

b31 b30 - b13		b12 - b8		b7 - b5	b4 - b0
1	-	Bit rate high-level register (ICBRH) value.		-	Bit rate low-level register (ICBRL) value.



**Description (3/3)****[data8]**

Rise and fall time compensation.

If the transfer rate is specified in bits per second, the high-level and low-level durations can be adjusted to allow for application-dependent rise and fall times. If unsure, use 0.

b31 - b16	b15 - b0
The SCL rise time in nanoseconds. Valid from 0 to 65535.	The SCL fall time in nanoseconds. Valid from 0 to 65535.

**Return value**

True if all parameters are valid, exclusive and achievable; otherwise false.

**Category**

I<sup>2</sup>C

**Reference**

R\_IIC\_Destroy

**Remarks**

- Function R\_CGC\_Set must be called before any use of this function.
- This function configures each I<sup>2</sup>C pin that is required for operation. It also disables the alternative modes on those pins.
- The 7 or 10-bit slave addresses should use the format:

b15 - b8	b7 - b1	b0
-	7-bit address	-

b15 - b11	b10 - b1	b0
-	10-bit address	-

- The timing limits depend on the frequency of the internal reference clock (IRC).

$$Transfer\_rate = \frac{1}{t_{rise} + t_{fall} + (ICBRH + 1)t_{IRC} + (ICBRL + 1)t_{IRC}}$$

The maximum transfer rate is given when ICBRH = ICBRL = 0; the minimum when ICBRH = ICBRL = 31.

The absolute limits (with zero rise and fall times) are:

f <sub>IRC</sub>	f <sub>PCLK</sub> (MHz)			
	50	12.5	32	8
f <sub>PCLK</sub> ÷ 1	781 kbps to 25.0 Mbps	195 kbps to 6.25 Mbps	500 kbps to 16.0 Mbps	125 kbps to 4.00 Mbps
f <sub>PCLK</sub> ÷ 2	391 kbps to 12.5 Mbps	97.7 kbps to 3.13 Mbps	250 kbps to 8.00 Mbps	62.5 kbps to 2.00 Mbps
f <sub>PCLK</sub> ÷ 4	195 kbps to 6.25 Mbps	48.8 kbps to 1.56 Mbps	125 kbps to 4.00 Mbps	31.3 kbps to 1.00 Mbps
f <sub>PCLK</sub> ÷ 8	97.7 kbps to 3.13 Mbps	24.4 kbps to 781 kbps	62.5 kbps to 2.00 Mbps	15.6 kbps to 500 kbps
f <sub>PCLK</sub> ÷ 16	48.8 kbps to 1.56 Mbps	12.2 kbps to 391 kbps	31.3 kbps to 1.00 Mbps	7.81 kbps to 250 kbps
f <sub>PCLK</sub> ÷ 32	24.4 kbps to 781 kbps	6.10 kbps to 195 kbps	15.6 kbps to 500 kbps	3.91 kbps to 125 kbps
f <sub>PCLK</sub> ÷ 64	12.2 kbps to 391 kbps	3.05 kbps to 97.7 kbps	7.81 kbps to 250 kbps	1.95 kbps to 62.5 kbps
f <sub>PCLK</sub> ÷ 128	6.10 kbps to 195 kbps	1.53 kbps to 48.8 kbps	3.91 kbps to 125 kbps	977 bps to 31.3 kbps

The actual rise and fall times will not be zero.

Using the limits from the I<sup>2</sup>C specification:

Rise time: (rate ≤ 100 kbps): 1000 ns; (100 kbps < rate ≤ 400 kbps): 300 ns; (400 kbps < rate ≤ 1 Mbps): 120 ns

Fall time: (rate ≤ 400 kbps): 300 ns; (400 kbps < rate ≤ 1 Mbps): 120 ns

Maximum rate: 1 Mbps

The achievable transfer rates are:

IRC	f <sub>PCLK</sub> (MHz)			
	50	12.5	32	8
PCLK ÷ 1	658 kbps to 1 Mbps	175 kbps to 1 Mbps	446 kbps to 1 Mbps	116 kbps to 1 Mbps
PCLK ÷ 2	336 kbps to 1 Mbps	86.7 kbps to 1 Mbps	217 kbps to 1 Mbps	57.8 kbps to 1 Mbps
PCLK ÷ 4	175 kbps to 1 Mbps	45.9 kbps to 1 Mbps	116 kbps to 1 Mbps	30.0 kbps to 806 kbps
PCLK ÷ 8	86.7 kbps to 1 Mbps	23.7 kbps to 658 kbps	57.8 kbps to 1 Mbps	15.3 kbps to 446 kbps
PCLK ÷ 16	45.9 kbps to 1 Mbps	12.0 kbps to 316 kbps	30.0 kbps to 806 kbps	7.73 kbps to 217 kbps
PCLK ÷ 32	23.7 kbps to 658 kbps	6.06 kbps to 175 kbps	15.3 kbps to 446 kbps	3.89 kbps to 116 kbps
PCLK ÷ 64	12.0 kbps to 316 kbps	3.04 kbps to 86.7 kbps	7.73 kbps to 217 kbps	1.95 kbps to 57.8 kbps
PCLK ÷ 128	6.06 kbps to 175 kbps	1.52 kbps to 45.9 kbps	3.89 kbps to 116 kbps	975 bps to 30.0 kbps

**Program example**

```
/* RPDL definitions */
#include "r_pdl_iic.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Select I2C mode at 100kHz, 100ns rise and fall times */
    R_IIC_Create(
        0,
        PDL_IIC_MODE_IIC | PDL_IIC_INT_PCLK_DIV_8,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        100E3,
        (100 << 16) | 100
    );

    /* Select I2C mode with two slave addresses */
    R_IIC_Create(
        0,
        PDL_IIC_MODE_IIC,
        PDL_IIC_SLAVE_0_ENABLE_7 | PDL_IIC_SLAVE_1_ENABLE_7,
        0x0020,
        0x0056,
        PDL_NO_DATA,
        100E3,
        (300 << 16) | 200
    );
}
```

## 2) R\_IIC\_Destroy

### Synopsis

Disable an I<sup>2</sup>C channel.

### Prototype

```
bool R_IIC_Destroy(
    uint8_t data    // Channel selection
);
```

### Description

Shut down the selected I<sup>2</sup>C module.

#### [data]

Select channel IICn (where n = 0).

### Return value

True if the parameter is valid; otherwise false.

### Category

I<sup>2</sup>C

### Reference

R\_IIC\_Create

### Remarks

- The I<sup>2</sup>C module is put into the power-down state.

### Program example

```
/* RPDL definitions */
#include "r_pdl_iic.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Shutdown IIC channel 0 */
    R_IIC_Destroy(
        0
    );
}
```

### 3) R\_IIC\_MasterSend

#### Synopsis

Write data to a slave device.

#### Prototype

```
bool R_IIC_MasterSend(
    uint8_t data1,    // Channel selection
    uint8_t data2,    // Channel configuration
    uint16_t data3,   // Slave address
    uint8_t * data4,  // Data start address
    uint16_t data5,   // Data count
    void * func,      // Callback function
    uint8_t data6     // Interrupt priority level
);
```

#### Description

Transmit data on the specified channel.

#### [data1]

Select channel IICn (where n = 0).

#### [data2]

Configure the channel. If multiple selections are required, use “|” to separate each selection. The default settings are shown in **bold**. Specify PDL\_NO\_DATA to use the defaults.

- Start / Repeated Start condition control

<b>PDL_IIC_START_ENABLE</b> or PDL_IIC_START_DISABLE	Choose whether or not to issue a Start or Repeated Start condition at the beginning of the transfer.
---	--

- Stop condition control

<b>PDL_IIC_STOP_ENABLE</b> or PDL_IIC_STOP_DISABLE	Choose whether or not to issue a Stop condition at the end of the transfer.
---	---

- Slave address size override

PDL_IIC_10_BIT_SLAVE_ADDRESS	Specify this option if 10-bit address mode is to be used instead of 7-bit mode when the slave address is ≤ FFh.
------------------------------	---

- DTC trigger control

<b>PDL_IIC_DTC_TRIGGER_DISABLE</b> or PDL_IIC_DTC_TRIGGER_ENABLE	Disable or enable activation of the DTC when a data byte is transmitted.
---	--

#### [data3]

The address of the slave device. Ignored if the Start condition is disabled.

#### [data4]

The start address of the data to be sent.

If the DTC shall be used to transfer the data, specify PDL\_NO\_PTR.

#### [data5]

The number of bytes to be sent.

If the DTC shall be used to transfer the data, specify PDL\_NO\_DATA.

#### [func]

Specify PDL\_NO\_FUNC or a callback function name, depending on the required transfer method.

Transfer method	Parameter
Polling	PDL_NO_FUNC. This function will continue until the required number of bytes has been sent (or another event occurs).
Interrupts	The function to be called when bus activity has stopped.
DTC	The function to be called at the interval specified in R_DTC_Create.

#### [data6]

The interrupt priority level. Select between 1 (lowest priority) and 15 (highest priority).

This parameter will be ignored if PDL\_NO\_FUNC is specified for parameter func.

<b>Return value</b>	True if all parameters are valid, exclusive and achievable and a normal transfer completed; otherwise false.
<b>Category</b>	I <sup>2</sup> C
<b>Reference</b>	R_DTC_Create, R_IIC_GetStatus, R_IIC_Control
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• If a callback function is specified, transmission interrupts are used. Please see the notes on callback function usage in §6.</li> <li>• If the Start condition is enabled and the previous transfer did not issue a Stop condition, a Repeated Start condition shall be generated.</li> <li>• If the Start condition is disabled, the slave address will not be transmitted.</li> <li>• If no callback function is specified for transmission completion, this function will monitor the status flags to manage the data transmission. If the I<sup>2</sup>C channel's registers are modified directly by the user, this function may lock up.</li> <li>• If false is returned, use R_IIC_GetStatus to check if an unexpected event on I<sup>2</sup>C bus was the cause of the failure. If the transfer has ended prematurely, use R_IIC_Control to issue a Stop condition.</li> </ul>

**Program example**

```

/* RPDL definitions */
#include "r_pdl_iic.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

const uint8_t data_array[5] = {0x23, 0x48, 0x59, 0x60, 0xFE};

void func(void)
{
    /* Send 5 bytes to device 0x0A0 on channel 0, using polling */
    R_IIC_MasterSend(
        0,
        PDL_NO_DATA,
        0x0A0,
        data_array,
        5,
        PDL_NO_FUNC,
        0
    );
}

```

## 4) R\_IIC\_MasterReceive

**Synopsis**

Read data from a slave device.

**Prototype**

```
bool R_IIC_MasterReceive(
    uint8_t data1,    // Channel selection
    uint8_t data2,    // Channel configuration
    uint16_t data3,   // Slave address
    uint8_t * data4,   // Data start address
    uint16_t data5,   // Receive threshold
    void * func,      // Callback function
    uint8_t data6     // Interrupt priority level
);
```

**Description**

Read data over an I<sup>2</sup>C channel and store it.

**[data1]**

Select channel IIC<sub>n</sub> (where n = 0).

**[data2]**

Configure the channel.

The default setting is shown in **bold**. Specify PDL\_NO\_DATA to use the defaults.

- Slave address size override

PDL_IIC_10_BIT_SLAVE_ADDRESS	Specify this option if 10-bit address mode is to be used instead of 7-bit mode when the slave address is ≤ FFh.
------------------------------	---

- DTC trigger control

<b>PDL_IIC_DTC_TRIGGER_DISABLE</b> or PDL_IIC_DTC_TRIGGER_ENABLE	Disable or enable activation of the DTC when a data byte is received.
---	---

**[data3]**

The address of the slave device.

**[data4]**

The start address of the storage area for the expected data.

Specify PDL\_NO\_PTR if no data shall be processed by this function e.g. if the DTC shall be used to process the received data.

**[data5]**

The number of bytes that must be received before the function completes or the callback function is called.

If the DTC shall be used to handle the received data, specify PDL\_NO\_DATA.

**[func]**

Specify PDL\_NO\_FUNC or a callback function name, depending on the required transfer method.

Transfer method	Parameter
Polling	PDL_NO_FUNC. This function will continue until the required number of bytes has been received (or another event occurs).
Interrupts	The function to be called when bus activity has stopped.
DTC	The function to be called at the interval specified in R_DTC_Create.

**[data6]**

The interrupt priority level. Select between 1 (lowest priority) and 15 (highest priority).

This parameter will be ignored if PDL\_NO\_FUNC is specified for parameter func.

**Return value**

True if all parameters are valid, exclusive and achievable; otherwise false.

**Category**

I<sup>2</sup>C

**Reference**

R\_IIC\_Create, R\_IIC\_GetStatus, R\_IIC\_Control

**Remarks**

- If a callback function is specified, reception interrupts are used. Please see the notes on callback function usage in §6.
- If the previous transfer did not issue a Stop condition, a Repeated Start condition shall be generated.
- The last byte to be read shall be completed with a NACK signal.
- If no callback function is specified, this function will operate in polling mode. The status flags will be used to manage the data reception. If the I<sup>2</sup>C channel's control registers are directly modified by the user, this function may lock up. If an error occurs during this polling process, the function will terminate.
- If the DTC is used, use R\_IIC\_MasterReceiveLast to complete the transfer.
- Use R\_IIC\_GetStatus to determine if the transfer was successful.

**Program example**

```

/* RPDL definitions */
#include "r_pdl_iic.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

volatile uint8_t data_array[5];

void func(void)
{
    /* Read 5 bytes from device 0xAA on channel 0, using polling */
    R_IIC_MasterReceive(
        0,
        PDL_NO_DATA,
        0xAA,
        data_array,
        5,
        PDL_NO_FUNC,
        0
    );
}

```

## 5) R\_IIC\_MasterReceiveLast

### Synopsis

Complete a DTC-based read process.

### Prototype

```
bool R_IIC_MasterReceiveLast(
    uint8_t data1,    // Channel selection
    uint8_t * data2   // Data storage address
);
```

### Description

Read one data byte with NACK and stop.

#### [data1]

Select channel IICn (where n = 0 or 1).

#### [data2]

The storage location for the data byte.

### Return value

True if all parameters are valid and the function completed; otherwise false.

### Category

I<sup>2</sup>C

### Reference

R\_IIC\_GetStatus

### Remarks

- This function must only be used to terminate a Read process that has used the DTC.
- Use R\_IIC\_GetStatus to determine if the transfer was successful.
- Please specify one byte less in the Transfer Count when using with the DTC.

### Program example

```
/* RPDL definitions */
#include "r_pdl_iic.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

volatile uint8_t data_array[5];

void func(void)
{
    /* Read 1 byte on channel 0 and stop */
    R_IIC_MasterReceiveLast(
        0,
        &data_array[4]
    );
}
```



## 6) R\_IIC\_SlaveMonitor

**Synopsis**

Monitor the bus.

**Prototype**

```
bool R_IIC_SlaveMonitor(
    uint8_t data1,    // Channel selection
    uint8_t data2,    // Channel configuration
    uint8_t * data3,  // Receive data start address
    uint16_t data4,   // Receive threshold
    void * func,      // Callback function
    uint8_t data5     // Interrupt priority level
);
```

**Description**

Monitor the bus until an address match occurs and store any data received.  
Register the storage area and transfer method for data received on the selected I<sup>2</sup>C channel.

**[data1]**

Select channel IIC<sub>n</sub> (where n = 0).

**[data2]**

Select the operation options.

The default setting is shown in **bold**. Specify PDL\_NO\_DATA to use the default.

- DTC trigger control

<b>PDL_IIC_RX_DTC_TRIGGER_DISABLE</b> or PDL_IIC_RX_DTC_TRIGGER_ENABLE	Disable or enable activation of the DTC when a byte is received.
<b>PDL_IIC_TX_DTC_TRIGGER_DISABLE</b> or PDL_IIC_TX_DTC_TRIGGER_ENABLE	Disable or enable activation of the DTC for data transmission.

**[data3]**

The start address of the storage area for any received data.

If the DTC shall be used to handle the received data, specify PDL\_NO\_PTR.

**[data4]**

The number of bytes in the storage area.

If the DTC shall be used to handle the received data, specify PDL\_NO\_DATA.

**[func]**

Specify PDL\_NO\_FUNC or a callback function name, depending on the required transfer method.

Transfer method	Parameter
Polling	PDL_NO_FUNC. This function will continue until a Stop condition is detected or the master tries to read data from this slave.
Interrupts	The function to be called when a Stop condition is detected or the master tries to read data from this slave.
DTC	The function to be called when a Stop or error condition is detected.

**[data5]**

The interrupt priority level. Select between 1 (lowest priority) and 15 (highest priority).

This parameter will be ignored if PDL\_NO\_FUNC is specified for parameter func.

**Return value**

True if all parameters are valid, exclusive and achievable; otherwise false.

**Category**

I<sup>2</sup>C

**Reference**

R\_IIC\_Create, R\_IIC\_GetStatus, R\_IIC\_SlaveSend

**Remarks**

- If a callback function is specified, interrupts are used. Use `R_IIC_GetStatus` in the callback function to identify the activity that has occurred. Please see the notes on callback function usage in §6.
- If no callback function is specified, this function will monitor the status flags to monitor the bus activity. Use `R_IIC_GetStatus` to identify the activity that has occurred. If the I<sup>2</sup>C channel's control registers are directly modified by the user, this function may lock up.
- If the master sends more data than is expected and the DTC trigger is disabled, this function will issue a NACK to the master.
- When a Stop condition is detected, if the DTC is used for transferring data, use `R_DTC_Control` to re-set the address and count before the next transfer begins.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_iic.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

volatile uint8_t data_array[5];

void func(void)
{
    /* Monitor channel 0, using polling */
    R_IIC_SlaveMonitor(
        0,
        PDL_NO_DATA,
        data_array,
        5,
        PDL_NO_FUNC,
        0
    );
}
```

## 7) R\_IIC\_SlaveSend

### Synopsis

Write data to a master device.

### Prototype

```
bool R_IIC_SlaveSend(
    uint8_t data1,    // Channel selection
    uint8_t * data2,  // Data start address
    uint16_t data3    // Data count
);
```

### Description

Transmit data on the specified channel.

#### [data1]

Select channel IICn (where n = 0).

#### [data2]

The start address of the data to be sent.

#### [data3]

The number of bytes available to be sent.

### Return value

True if all parameters are valid, exclusive and achievable; otherwise false.  
If this function is not called from the R\_IIC\_SlaveMonitor callback function, it will complete when a stop condition is detected.

### Category

I<sup>2</sup>C

### Reference

R\_IIC\_SlaveMonitor

### Remarks

- Use this function in conjunction with R\_IIC\_SlaveMonitor.
- If the master requires more data than is supplied, and polling or interrupt-based transfers are used, this function shall loop back to the start of the data. The transmitted byte count will also be reset to 0.

### Program example

```
/* RPDL definitions */
#include "r_pdl_iic.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

const uint8_t data_array[5] = {0x23, 0x48, 0x59, 0x60, 0xFE};

void func(void)
{
    /* Assign 5 bytes to be read by a master on channel 0 */
    R_IIC_SlaveSend(
        0,
        data_array,
        5
    );
}
```

## 8) R\_IIC\_Control

### Synopsis

I<sup>2</sup>C channel control.

### Prototype

```
bool R_IIC_Control(
    uint8_t data1,    // Channel selection
    uint8_t data2     // Control options
);
```

### Description

Modify the operation of the selected I<sup>2</sup>C channel.

#### [data1]

Select channel IIC<sub>n</sub> (where n = 0).

#### [data2]

Control the channel. If multiple selections are required, use “|” to separate each selection.

- Stop generation

PDL_IIC_STOP	Issue a Stop condition.
--------------	-------------------------

- NACK generation

PDL_IIC_NACK	Set the Acknowledge bit to the NACK state.
--------------	--

- Pin control

PDL_IIC_SDA_LOW or PDL_IIC_SDA_HI_Z	Set the SDA pin to low level or high-impedance.
--	---

PDL_IIC_SCL_LOW or PDL_IIC_SCL_HI_Z	Set the SCL pin to low level or high-impedance.
--	---

- Extra clock cycle generation

PDL_IIC_CYCLE_SCL	Generate an extra clock cycle on the SCL pin. This can be used in Master mode to try and unlock a slave device that is holding the SDA signal low.
-------------------	--

- Reset control

PDL_IIC_RESET	Carry out an internal reset of the I <sup>2</sup> C module (the settings are preserved).
---------------	--

### Return value

True if all parameters are valid, exclusive and achievable; otherwise false.

### Category

I<sup>2</sup>C

### Reference

R\_IIC\_Create

### Remarks

- None.

### Program example

```
/* RPDL definitions */
#include "r_pdl_iic.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Issue a Stop condition on channel 0 */
    R_IIC_Control(
        0,
        PDL_IIC_STOP
    );
}
```

## 9) R\_IIC\_GetStatus

### Synopsis

Read the status for an I<sup>2</sup>C channel.

### Prototype

```
bool R_IIC_GetStatus(
    uint8_t data1,    // Channel selection
    uint32_t * data2, // Status flags
    uint16_t * data3, // Transmitted bytes
    uint16_t * data4  // Received bytes
);
```

### Description

Read the status registers for the selected I<sup>2</sup>C channel.

#### [data1]

Select channel IIC<sub>n</sub> (where n = 0).

#### [data2]

The status flags shall be stored in the format below.  
Specify PDL\_NO\_PTR if this information is not required.

b31 – b18								b17	b16
0								Buffer status	
								Transmit	Receive
								0: Full 1: Empty	0: Empty 1: Full

b15	b14	b13	b12	b11	b10	b9	b8
Bus state	Pin level		Event detection (0 = Not detected, 1 = detected)				
0: Idle 1: Busy	SCL	SDA	NACK	Stop condition	Start condition	Arbitration lost	Timeout

b7	b6	b5	b4	b3	b2	b1	b0
Transmission	Mode	Address detection (0 = Not detected, 1 = detected)					
0: Active 1: Idle	0: Receive 1: Transmit	SMBus host	Device-ID	General call	Slave		
					2	1	0

#### [data3]

The address for storing the number of bytes that are have been transmitted in the current transfer.  
Specify PDL\_NO\_PTR if this information is not required.

#### [data4]

The address for storing for the number of bytes that are have been received in the current transfer. Specify PDL\_NO\_PTR if this information is not required.

### Return value

True if all parameters are valid; otherwise false.

### Category

I<sup>2</sup>C

### Reference

R\_IIC\_Create

### Remarks

- The flags are not modified by this function. The event detection flags are cleared when a new transfer is started.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_iic.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    uint32_t status_flags;
    uint16_t tx_count;

    /* Read the status of channel 0 */
    R_IIC_GetStatus(
        0,
        &status_flags,
        tx_count,
        PDL_NO_PTR
    );
}
```

## 4.2.19. Serial Peripheral Interface

## 1) R\_SPI\_Create

**Synopsis**

Configure an SPI channel.

**Prototype**

```
bool R_SPI_Create(
    uint8_t data1, // Channel selection
    uint32_t data2, // Channel configuration
    uint32_t data3, // Data format
    uint32_t data4, // Extended timing control
    uint32_t data5 // Bit rate or register value
);
```

**Description (1/3)**

Set up the selected SPI channel.

**[data1]**

Select channel SPIn (where n = 0 only).

**[data2]**

Configure the channel mode and connection settings.

If multiple selections are required, use “|” to separate each selection.

The default settings are shown in **bold**.

## • Connection mode

PDL\_SPI\_MODE\_SPI\_MASTER or  
PDL\_SPI\_MODE\_SPI\_MULTI\_MASTER or  
PDL\_SPI\_MODE\_SPI\_SLAVE or  
PDL\_SPI\_MODE\_SYNC\_MASTER or  
PDL\_SPI\_MODE\_SYNC\_SLAVE

The required SPI (four-wire) or Clock  
synchronous (three-wire operation)  
connection type.

## • Reception control

**PDL\_SPI\_FULL\_DUPLEX** or  
PDL\_SPI\_TRANSMIT\_ONLY

Enable or disable reception operations.

## • Pin selection and control

PDL\_SPI\_PIN\_A or  
PDL\_SPI\_PIN\_B or  
PDL\_SPI\_PIN\_C

Select the -A, -B or -C pins for signals MISO,  
MOSI, RSPCK, SSL0, SSL1, SSL2 and SSL3.

**PDL\_SPI\_PIN\_RSPCK\_ENABLE** or  
PDL\_SPI\_PIN\_RSPCK\_DISABLE

Enable or disable signal RSPCK.

**PDL\_SPI\_PIN\_MOSI\_ENABLE** or  
PDL\_SPI\_PIN\_MOSI\_DISABLE

Enable or disable output signal MOSI.

**PDL\_SPI\_PIN\_MISO\_ENABLE** or  
PDL\_SPI\_PIN\_MISO\_DISABLE

Enable or disable input signal MISO.

PDL\_SPI\_PIN\_SSL0\_LOW or  
PDL\_SPI\_PIN\_SSL0\_HIGH or  
**PDL\_SPI\_PIN\_SSL0\_DISABLE**

Select active-low,  
active-high or  
disabled for output signal SSL0.

PDL\_SPI\_PIN\_SSL1\_LOW or  
PDL\_SPI\_PIN\_SSL1\_HIGH or  
**PDL\_SPI\_PIN\_SSL1\_DISABLE**

Select active-low,  
active-high or  
disabled for output signal SSL1.

PDL\_SPI\_PIN\_SSL2\_LOW or  
PDL\_SPI\_PIN\_SSL2\_HIGH or  
**PDL\_SPI\_PIN\_SSL2\_DISABLE**

Select active-low,  
active-high or  
disabled for output signal SSL2.

PDL\_SPI\_PIN\_SSL3\_LOW or  
PDL\_SPI\_PIN\_SSL3\_HIGH or  
**PDL\_SPI\_PIN\_SSL3\_DISABLE**

Select active-low,  
active-high or  
disabled for output signal SSL3.

**PDL\_SPI\_PIN\_MOSI\_IDLE\_LAST** or  
PDL\_SPI\_PIN\_MOSI\_IDLE\_LOW or  
PDL\_SPI\_PIN\_MOSI\_IDLE\_HIGH

The MOSI output state when no SSLn pin is  
active.

**Description (2/3)****[data3]**

Configure the data format. If multiple selections are required, use “|” to separate each selection. The default settings are shown in **bold**.

- Buffer size

<b>PDL_SPI_BUFFER_64</b> or <b>PDL_SPI_BUFFER_128</b>	Select a buffer size of 64 bits (up to four 16-bit frames) or 128 bits (up to four 32-bit frames).
---	--

- Frame configuration selection (refer to figure 25.2 in the hardware manual).

Selection	Number of command transfers	Number of frames in each command transfer	Total number of transfer frames
PDL_SPI_FRAME_1_1 or PDL_SPI_FRAME_1_2 or PDL_SPI_FRAME_1_3 or PDL_SPI_FRAME_1_4 or PDL_SPI_FRAME_2_1 or PDL_SPI_FRAME_2_2 or PDL_SPI_FRAME_3 or PDL_SPI_FRAME_4 or PDL_SPI_FRAME_5 or PDL_SPI_FRAME_6 or PDL_SPI_FRAME_7 or PDL_SPI_FRAME_8	1 1 1 1 2 2 3 4 5 6 7 8	1 2 3 4 1 2 1 1 1 1 1 1	1 2 3 4 2 4 3 4 5 6 7 8

- Parity bit control

<b>PDL_SPI_PARITY_NONE</b> or PDL_SPI_PARITY_EVEN or PDL_SPI_PARITY_ODD	Disable or enable the addition of the parity bit.
---	---

**[data4]**

Extended timing control (optional).

All items apply only to Master mode.

If multiple selections are required, use “|” to separate each selection.

The default settings are shown in **bold**. Specify PDL\_NO\_DATA if not required.

- Extended clock delay

<b>PDL_SPI_CLOCK_DELAY_1</b> or PDL_SPI_CLOCK_DELAY_2 or PDL_SPI_CLOCK_DELAY_3 or PDL_SPI_CLOCK_DELAY_4 or PDL_SPI_CLOCK_DELAY_5 or PDL_SPI_CLOCK_DELAY_6 or PDL_SPI_CLOCK_DELAY_7 or PDL_SPI_CLOCK_DELAY_8	The number of bit clock periods between the assertion of the SSL pin and the start of RSPCK oscillation. Ignored in Slave mode.
---	---

- Extended SSL negation delay

<b>PDL_SPI_SSL_DELAY_1</b> or PDL_SPI_SSL_DELAY_2 or PDL_SPI_SSL_DELAY_3 or PDL_SPI_SSL_DELAY_4 or PDL_SPI_SSL_DELAY_5 or PDL_SPI_SSL_DELAY_6 or PDL_SPI_SSL_DELAY_7 or PDL_SPI_SSL_DELAY_8	The number of bit clock periods between the end of RSPCK oscillation and the negation of the active SSL pin. Ignored in Slave mode.
---	---

- Extended next-access delay

<b>PDL_SPI_NEXT_DELAY_1</b> or PDL_SPI_NEXT_DELAY_2 or PDL_SPI_NEXT_DELAY_3 or PDL_SPI_NEXT_DELAY_4 or PDL_SPI_NEXT_DELAY_5 or PDL_SPI_NEXT_DELAY_6 or PDL_SPI_NEXT_DELAY_7 or PDL_SPI_NEXT_DELAY_8	The number of bit clock periods (plus two cycles of the peripheral clock) between the end of one frame and the start of the next frame. Ignored in Slave mode.
---	--



Description (3/3)

[data5]

The format must be either:

The maximum required bit rate.

Or:

b31

b30 to b8

b7 – b0

1

0

The SPBR register value.

If only Slave mode will be used, specify PDL\_NO\_DATA.

Return value

True if all parameters are valid; otherwise false.

Category

SPI

Reference

R\_CGC\_Set, R\_SPI\_Command

Remarks

Function R\_CGC\_Set must be called before any use of this function.

-C pin selection is not available on the 80-pin and 64-pin packages.

SSL2-B pin and SSL3-B pin cannot be selected on the 80-pin and 64-pin packages.

The actual bit rate will be reduced if division > 1 is specified in R\_SPI\_Command.

Program example

```

/* RPDL definitions */
#include "r_pdl_spi.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Configure SPI channel 0 */
    R_SPI_Create(
        0,
        PDL_SPI_MODE_SPI_MASTER | PDL_SPI_PIN_SSL0_LOW | PDL_SPI_PIN_A,
        PDL_SPI_FRAME_1_1,
        PDL_NO_DATA,
        2E6
    );
}

```

## 2) R\_SPI\_Destroy

**Synopsis**

Shutdown an SPI channel.

**Prototype**

```
bool R_SPI_Destroy(  
    uint8_t data    // Channel selection  
);
```

**Description**

Shutdown the selected SPI channel.

**[data]**

Select channel SPIn (where n = 0 only).

**Return value**

True if all parameters are valid; otherwise false.

**Category**

SPI

**Reference**

R\_SPI\_Create

**Remarks**

- The SPI channel is put into the power-down state.

**Program example**

```
/* RPDL definitions */  
#include "r_pdl_spi.h"  
  
/* RPDL device-specific definitions */  
#include "r_pdl_definitions.h"  
  
void func(void)  
{  
    /* Shutdown SPI channel 0 */  
    R_SPI_Destroy(  
        0  
    );  
}
```

### 3) R\_SPI\_Command

#### Synopsis

Configure an SPI command.

#### Prototype

```
bool R_SPI_Command(
    uint8_t data1,    // Channel selection
    uint8_t data2,    // Command selection
    uint32_t data3,   // Command options
    uint8_t data4     // Extended timing control
);
```

#### Description (1/2)

Select the options for a command.

##### [data1]

Select channel SPIn (where n = 0 only).

##### [data2]

Select command n (where n = 0 to 7).

##### [data3]

Select the command options. If multiple selections are required, use “|” to separate each selection.

The default settings are shown in **bold**.

- Clock phase and polarity

PDL_SPI_CLOCK_MODE_0 or PDL_SPI_CLOCK_MODE_1 or PDL_SPI_CLOCK_MODE_2 or	Clock is low when idle; data is sampled on the rising edge. Clock is low when idle; data is sampled on the falling edge. Clock is high when idle; data is sampled on the falling edge.
PDL_SPI_CLOCK_MODE_3	Clock is high when idle; data is sampled on the rising edge.

- Clock division

<b>PDL_SPI_DIV_1</b> or PDL_SPI_DIV_2 or PDL_SPI_DIV_4 or PDL_SPI_DIV_8	Use the bit rate (specified for R_SPI_Create) ÷ 1, 2, 4 or 8. Ignored in Slave mode.
--	---

- SSL assertion

<b>PDL_SPI_ASSERT_SSL0</b> or PDL_SPI_ASSERT_SSL1 or PDL_SPI_ASSERT_SSL2 or PDL_SPI_ASSERT_SSL3	The SSL pin to be asserted during the frame transfer. Ignored in Slave mode.
--	---

- SSL negation

<b>PDL_SPI_SSL_NEGATE</b> or PDL_SPI_SSL_KEEP	Negate or retain the SSL signal after the frame transfer. Ignored in Slave mode.
--	---

- Frame data length

PDL_SPI_LENGTH_8 or PDL_SPI_LENGTH_9 or PDL_SPI_LENGTH_10 or PDL_SPI_LENGTH_11 or PDL_SPI_LENGTH_12 or PDL_SPI_LENGTH_13 or PDL_SPI_LENGTH_14 or PDL_SPI_LENGTH_15 or PDL_SPI_LENGTH_16 or PDL_SPI_LENGTH_20 or PDL_SPI_LENGTH_24 or PDL_SPI_LENGTH_32	The number of bits in the frame transfer. If a buffer size of 64 bits was selected when R_SPI_Create was called, the number of bits must not exceed 16.
---	--

- Data transfer format

PDL_SPI_MSB_FIRST or PDL_SPI_LSB_FIRST	Select least- or most-significant bit first.
---	--

<b>Description (2/2)</b>	<p><b>[data4]</b> Extended timing control. If multiple selections are required, use “ ” to separate each selection. The default settings are shown in <b>bold</b>. For Slave mode, select PDL_NO_DATA.</p> <ul style="list-style-type: none"> <li>Extended timing selection <table border="1"> <tr> <td><b>PDL_SPI_CLOCK_DELAY_MINIMUM</b> or <b>PDL_SPI_CLOCK_DELAY_EXTENDED</b></td><td>Select the minimum or extended delay between the assertion of the SSL pin and the start of RSPCK oscillation.</td></tr> </table> </li> <li>SSL negation delay <table border="1"> <tr> <td><b>PDL_SPI_SSL_DELAY_MINIMUM</b> or <b>PDL_SPI_SSL_DELAY_EXTENDED</b></td><td>Select the minimum or extended delay between the end of RSPCK oscillation and the negation of the active SSL pin.</td></tr> </table> </li> <li>Next-access delay <table border="1"> <tr> <td><b>PDL_SPI_NEXT_DELAY_MINIMUM</b> or <b>PDL_SPI_NEXT_DELAY_EXTENDED</b></td><td>Select the minimum or extended delay between the end of one frame and the start of the next frame.</td></tr> </table> </li> </ul>	<b>PDL_SPI_CLOCK_DELAY_MINIMUM</b> or <b>PDL_SPI_CLOCK_DELAY_EXTENDED</b>	Select the minimum or extended delay between the assertion of the SSL pin and the start of RSPCK oscillation.	<b>PDL_SPI_SSL_DELAY_MINIMUM</b> or <b>PDL_SPI_SSL_DELAY_EXTENDED</b>	Select the minimum or extended delay between the end of RSPCK oscillation and the negation of the active SSL pin.	<b>PDL_SPI_NEXT_DELAY_MINIMUM</b> or <b>PDL_SPI_NEXT_DELAY_EXTENDED</b>	Select the minimum or extended delay between the end of one frame and the start of the next frame.
<b>PDL_SPI_CLOCK_DELAY_MINIMUM</b> or <b>PDL_SPI_CLOCK_DELAY_EXTENDED</b>	Select the minimum or extended delay between the assertion of the SSL pin and the start of RSPCK oscillation.						
<b>PDL_SPI_SSL_DELAY_MINIMUM</b> or <b>PDL_SPI_SSL_DELAY_EXTENDED</b>	Select the minimum or extended delay between the end of RSPCK oscillation and the negation of the active SSL pin.						
<b>PDL_SPI_NEXT_DELAY_MINIMUM</b> or <b>PDL_SPI_NEXT_DELAY_EXTENDED</b>	Select the minimum or extended delay between the end of one frame and the start of the next frame.						
<b>Return value</b>	True if all parameters are valid; otherwise false.						
<b>Category</b>	SPI						
<b>Reference</b>	R_SPI_Create						
<b>Remarks</b>	<ul style="list-style-type: none"> <li>For Slave mode operation, configure command 0.</li> <li>When Clock-synchronous Slave mode is used, avoid selecting mode 0 or mode 2.</li> <li>If parity is enabled while in Master mode, both the frame data length and data transfer format should be the same for each command.</li> </ul>						

**Program example**

```

/* RPDL definitions */
#include "r_pdl_spi.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Configure SPI channel 0 commands 0 and 1 */
    R_SPI_Command(
        0,
        0,
        PDL_SPI_CLOCK_MODE_0 | PDL_SPI_ASSERT_SSL0 | \
        PDL_SPI_LENGTH_8 | PDL_SPI_MSB_FIRST,
        PDL_NO_DATA
    );

    R_SPI_Command(
        0,
        1,
        PDL_SPI_CLOCK_MODE_1 | PDL_SPI_ASSERT_SSL1 | \
        PDL_SPI_LENGTH_8 | PDL_SPI_LSB_FIRST,
        PDL_NO_DATA
    );
}

```

#### 4) R\_SPI\_Transfer

##### Synopsis

Transfer data over an SPI channel.

##### Prototype

```
bool R_SPI_Transfer(
    uint8_t data1,    // Channel selection
    uint8_t data2,    // DTC control
    uint32_t * data3, // Transmit data start address
    uint32_t * data4, // Receive data start address
    uint16_t data5,   // Sequence loop count
    void * func,      // Callback function
    uint8_t data6     // Interrupt priority level
);
```

##### Description

In Master mode, transfer the data to and/or from the Slave device.  
In Slave mode, transfer the data under control of the Master device.

##### [data1]

Select channel SPIn (where n = 0 only).

##### [data2]

Select the automatic data transfer options.

The default setting is shown in **bold**. Specify PDL\_NO\_DATA to use the default.

- DTC trigger control

<b>PDL_SPI_DTC_TRIGGER_DISABLE</b> or PDL_SPI_DTC_TRIGGER_ENABLE	Disable or enable activation of the DTC for data transmission and reception.
---	--

##### [data3]

The start address of the data to be transmitted. The data must be stored as 32-bit values.

Specify PDL\_NO\_PTR if no data is to be transmitted (or if the data content is not important), or if the DTC shall be used to handle the data transfer.

##### [data4]

The start address of the data to be received. The data will be stored as 32-bit values.

Specify PDL\_NO\_PTR if no data is to be received, or if the DTC shall be used to handle the data transfer.

##### [data5]

The number of times that the command sequence will be executed.

If the DTC shall be used to handle the transfer, specify PDL\_NO\_DATA.

##### [func]

Specify PDL\_NO\_FUNC or a callback function name, depending on the required transfer method.

Transfer method	Parameter
Polling	PDL_NO_FUNC. This function will handle the data transfer until completion or an error occurs.
Interrupts	The function to be called when the transfer has completed or an error has occurred.
DTC	The function to be called if an error has occurred, or when the DTC passes on the transfer interrupt.

##### [data6]

The interrupt priority level. Select between 1 (lowest priority) and 15 (highest priority).

This parameter will be ignored if PDL\_NO\_FUNC is specified for parameter func.

##### Return value

True if all parameters are valid; otherwise false.

##### Category

SPI

##### Reference

R\_SPI\_Create, R\_SPI\_GetStatus

**Remarks**

- The amount of data for must match the total number of transfer frames (refer to parameter data3 in R\_SPI\_Create).
- If a callback function is specified and DTC control is not used, interrupts are used to handle the data transfer.  
Please see the notes on callback function usage in §6.
- When using transmit only in slave mode, return of the function by using polling or trigger of interrupt by using interrupt or DTC / DMAC does not ensure the end of transmission.
- After using this function, use R\_SPI\_GetStatus to check for and clear any error flags.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_spi.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    uint32_t transmit_data[8];
    uint32_t receive_data[8];

    /* Transmit and receive all enabled frames once */
    R_SPI_Transfer(
        0,
        PDL_NO_DATA,
        transmit_data,
        receive_data,
        1,
        PDL_NO_FUNC,
        0
    );
}
```

## 5) R\_SPI\_Control

### Synopsis

Control an SPI channel.

### Prototype

```
bool R_SPI_Control(
    uint8_t data1,    // Channel selection
    uint8_t data2,    // Control options
    uint32_t data3    // Extended timing control
);
```

### Description

Modify the operation of the selected SPI channel.

#### [data1]

Select channel SPIn (where n = 0 only).

#### [data2]

Control the channel. If multiple selections are required, use “|” to separate each selection. All items are optional. Specify PDL\_NO\_DATA if not required.

- Channel control

PDL_SPI_DISABLE	Disable and partially initialise the SPI channel.
-----------------	---

- Loopback control

PDL_SPI_LOOPBACK_DISABLE or PDL_SPI_LOOPBACK_DIRECT or PDL_SPI_LOOPBACK_REVERSED	Disable or enable loopback in direct or reversed mode.
--	--

#### [data3]

Extended timing control (optional).

All items apply only to Master mode. Specify PDL\_NO\_DATA if not required.

If multiple selections are required, use “|” to separate each selection.

- Extended clock delay

PDL_SPI_CLOCK_DELAY_1 or PDL_SPI_CLOCK_DELAY_2 or PDL_SPI_CLOCK_DELAY_3 or PDL_SPI_CLOCK_DELAY_4 or PDL_SPI_CLOCK_DELAY_5 or PDL_SPI_CLOCK_DELAY_6 or PDL_SPI_CLOCK_DELAY_7 or PDL_SPI_CLOCK_DELAY_8	The number of bit clock periods between the assertion of the SSL pin and the start of RSPCK oscillation. Ignored in Slave mode.
---	---

- Extended SSL negation delay

PDL_SPI_SSL_DELAY_1 or PDL_SPI_SSL_DELAY_2 or PDL_SPI_SSL_DELAY_3 or PDL_SPI_SSL_DELAY_4 or PDL_SPI_SSL_DELAY_5 or PDL_SPI_SSL_DELAY_6 or PDL_SPI_SSL_DELAY_7 or PDL_SPI_SSL_DELAY_8	The number of bit clock periods between the end of RSPCK oscillation and the negation of the active SSL pin. Ignored in Slave mode.
---	---

- Extended next-access delay

PDL_SPI_NEXT_DELAY_1 or PDL_SPI_NEXT_DELAY_2 or PDL_SPI_NEXT_DELAY_3 or PDL_SPI_NEXT_DELAY_4 or PDL_SPI_NEXT_DELAY_5 or PDL_SPI_NEXT_DELAY_6 or PDL_SPI_NEXT_DELAY_7 or PDL_SPI_NEXT_DELAY_8	The number of bit clock periods (plus two cycles of the peripheral clock) between the end of one frame and the start of the next frame. Ignored in Slave mode.
---	--

### Return value

True if all parameters are valid; otherwise false.

<b>Category</b>	SPI
<b>Reference</b>	R_SPI_Create
<b>Remarks</b>	<ul style="list-style-type: none"> <li>If a channel is disabled using PDL_SPI_DISABLE, call R_SPI_Create to resume channel operations.</li> </ul>
<b>Program example</b>	<pre> /* RPDL definitions */ #include "r_pdl_spi.h"  /* RPDL device-specific definitions */ #include "r_pdl_definitions.h"  void func(void) {     /* Enable direct loopback mode */     R_SPI_Control(         0,         PDL_SPI_LOOPBACK_DIRECT,         PDL_NO_DATA     );      /* Change the extended timings */     R_SPI_Control(         0,         PDL_NO_DATA,         PDL_SPI_CLOCK_DELAY_8   PDL_SPI_SSL_DELAY_5     ); } </pre>



## 6) R\_SPI\_GetStatus

### Synopsis

Check the status of an SPI channel.

### Prototype

```
bool R_SPI_GetStatus(
    uint8_t data1,    // Channel selection
    uint16_t * data2, // Status flags
    uint16_t * data3  // Sequence count
);
```

### Description

Acquires the SPI channel status.

#### [data1]

Select channel SPIn (where n = 0 only).

#### [data2]

The status flags shall be stored in the format below.

Specify PDL\_NO\_PTR if this information is not required

b15	b14 – b12	b11	b10 – b8
0	Error command	0	Command pointer

b7	b6	b5	b4	b3	b2	b1	b0
Receive buffer	0	Transmit buffer	0	Parity error	Mode fault	Bus state	Overrun error
0: Empty 1: Full		0: Full 1: Empty		0: No error 1: Detected	0: No fault 1: Detected	0: Idle 1: Active	0: No error 1: Detected

#### [data3]

The storage location for the number of sequence loops that have been completed in the current transfer. Specify PDL\_NO\_PTR if this information is not required.

### Return value

True if all parameters are valid; otherwise false.

### Category

SPI

### Reference

### Remarks

- If the status flags are read and an error or fault flag is set to 1, the flag will be cleared to 0 by this function.

### Program example

```
/* RPDL definitions */
#include "r_pdl_spi.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    uint16_t StatusValue;

    /* Read the status of channel 0 */
    R_SPI_GetStatus(
        0,
        &StatusValue,
        PDL_NO_PTR
    );
}
```

## 4.2.20. LIN module

## 1) R\_LIN\_Create

**Synopsis**

Configure the LIN channel.

**Prototype**

```

bool R_LIN_Create(
    uint8_t data1,    // Channel selection
    uint8_t data2,    // Channel configuration
    uint32_t data3,   // Transfer Break timing configuration
    uint32_t data4,   // Interbyte and Wake-up timing configuration
    uint32_t data5,   // Bit rate or register value
    void * func,      // Callback function
    uint8_t data6     // Interrupt priority level
);

```

**Description (1/2)**

Set up the selected LIN channel.

**[data1]**

Select LIN channel n (where n = 0).

**[data2]**

Configure the error detection settings.

If multiple selections are required, use "|" to separate each selection.

The default settings are shown in **bold**.

- Error detection control

<b>PDL_LIN_ERROR_BIT_DISABLE</b> or PDL_LIN_ERROR_BIT_ENABLE	Disable or enable bit error detection.
<b>PDL_LIN_ERROR_BUS_DISABLE</b> or PDL_LIN_ERROR_BUS_ENABLE	Disable or enable physical bus error detection.
<b>PDL_LIN_ERROR_FRAME_TIMEOUT_DISABLE</b> or PDL_LIN_ERROR_FRAME_TIMEOUT_ENABLE	Disable or enable frame timeout error detection.
<b>PDL_LIN_ERROR_FRAMING_DISABLE</b> or PDL_LIN_ERROR_FRAMING_ENABLE	Disable or enable framing error detection.
<b>PDL_LIN_ERROR_ALL_ENABLE</b>	Enable all error detection.

**[data3]**

Configure the Transmit Break timing settings. Use "|" to separate each selection.

- Transmit Break (Low) width

PDL_LIN_TB_LOW_13 or PDL_LIN_TB_LOW_14 or PDL_LIN_TB_LOW_15 or PDL_LIN_TB_LOW_16 or PDL_LIN_TB_LOW_17 or PDL_LIN_TB_LOW_18 or PDL_LIN_TB_LOW_19 or PDL_LIN_TB_LOW_20 or PDL_LIN_TB_LOW_21 or PDL_LIN_TB_LOW_22 or PDL_LIN_TB_LOW_23 or PDL_LIN_TB_LOW_24 or PDL_LIN_TB_LOW_25 or PDL_LIN_TB_LOW_26 or PDL_LIN_TB_LOW_27 or PDL_LIN_TB_LOW_28	The number of Tbits used for the length of the break pulse in the transmit frame header.
---	--

- Transmit Break Delimiter (High) width

PDL_LIN_TB_HIGH_1 or PDL_LIN_TB_HIGH_2 or PDL_LIN_TB_HIGH_3 or PDL_LIN_TB_HIGH_4	The number of Tbits used for the length of the break delimiter in the transmit frame header.
---	--

**Description (2/2)****[data4]**

Configure the Interbyte and Wake-up timing settings. Use “|” to separate each selection.

- Interbyte and response space width

PDL_LIN_IBHR_SPACE_0 or PDL_LIN_IBHR_SPACE_1 or PDL_LIN_IBHR_SPACE_2 or PDL_LIN_IBHR_SPACE_3 or PDL_LIN_IBHR_SPACE_4 or PDL_LIN_IBHR_SPACE_5 or PDL_LIN_IBHR_SPACE_6 or PDL_LIN_IBHR_SPACE_7	The number of Tbits used for the length of the interbyte (header) and response space in the transmit frame header.
---	--

- Interbyte space width

PDL_LIN_IB_SPACE_0 or PDL_LIN_IB_SPACE_1 or PDL_LIN_IB_SPACE_2 or PDL_LIN_IB_SPACE_3	The number of Tbits used for the length of the interbyte space in the transmit frame response.
---	--

- Wake-up detection control

PDL_LIN_WAKE_SPEC_13 or PDL_LIN_WAKE_SPEC_2	Use the wake-up detection timing to suit LIN specification 1.3 or 2.0 / 2.1.
--	--

- Wake-up pulse transmission width

PDL_LIN_WAKE_LENGTH_1 or PDL_LIN_WAKE_LENGTH_2 or PDL_LIN_WAKE_LENGTH_3 or PDL_LIN_WAKE_LENGTH_4 or PDL_LIN_WAKE_LENGTH_5 or PDL_LIN_WAKE_LENGTH_6 or PDL_LIN_WAKE_LENGTH_7 or PDL_LIN_WAKE_LENGTH_8 or PDL_LIN_WAKE_LENGTH_9 or PDL_LIN_WAKE_LENGTH_10 or PDL_LIN_WAKE_LENGTH_11 or PDL_LIN_WAKE_LENGTH_12 or PDL_LIN_WAKE_LENGTH_13 or PDL_LIN_WAKE_LENGTH_14 or PDL_LIN_WAKE_LENGTH_15 or PDL_LIN_WAKE_LENGTH_16	The number of Tbits used for the length of the transmitted wake-up pulse.
--	---

**[data5]**

The format must be one of the following:

- The required bit rate. The range is from 1k to 20k bps.

- Or register values in the following format

b31	b30 to b18	b17 - b16	b15 - b8	b7 - b0
1	0	LCKS	LBRP1	LBRP0

**[func]**

The function to be called when an event occurs. The valid events are:

- d) Data or Wake-up transmission completion
- e) Data or Wake-up reception completion
- f) Error detection

If not required, specify PDL\_NO\_FUNC.

**[data6]**

The interrupt priority level. Select between 1 (lowest priority) and 15 (highest priority).

This parameter will be ignored if PDL\_NO\_FUNC is specified for parameter func.

**Return value**

True if all parameters are valid and exclusive; otherwise false.

**Category**

LIN module

<b>Reference</b>	R_CGC_Set, R_LIN_Transfer, R_LIN_Read
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Function R_CGC_Set must be called before any use of this function.</li> <li>• This function puts the LIN module into Operation mode.</li> <li>• One Tbit is the transfer bit period.</li> <li>• The timing width settings allow the maximum frame timeout period to be exceeded. Please refer to the value of TFRAME_MAX in the appropriate version of the LIN specification.</li> <li>• If a callback function is specified, interrupts are used. Please see the notes on callback function usage in §6.</li> <li>• A callback function is executed by the interrupt processing function. This means that no other interrupt can be processed until the callback function has completed.</li> <li>• Read the received data using R_LIN_Read.</li> </ul>

**Program example**

```

/* RPDL definitions */
#include "r_pdl_lin.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

/* LIN event handler */
void LIN_Event_handler(void){}

void func(void)
{
    /* Configure LIN channel 0 */
    R_LIN_Create(
        0,
        PDL_LIN_ERROR_ALL_ENABLE,
        PDL_LIN_TB_LOW_13 | PDL_LIN_TB_HIGH_1,
        PDL_LIN_IBHR_SPACE_0 | PDL_LIN_IB_SPACE_0 | \
        PDL_LIN_WAKE_SPEC_2 | PDL_LIN_WAKE_LENGTH_1,
        19200,
        LIN_Event_handler,
        10
    );
}

```

## 2) R\_LIN\_Destroy

### Synopsis

Shutdown the LIN channel.

### Prototype

```
bool R_LIN_Destroy(
    uint8_t data    // Channel selection
);
```

### Description

Shutdown the selected LIN channel.

#### [data]

Select LIN channel n (where n = 0).

### Return value

True if all parameters are valid; otherwise false.

### Category

LIN module

### Reference

R\_LIN\_Create

### Remarks

- The LIN module is put into the power-down state.

### Program example

```
/* RPDL definitions */
#include "r_pdl_lin.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Shutdown the LIN channel */
    R_LIN_Destroy(
        0
    );
}
```

### 3) R\_LIN\_Transfer

#### Synopsis

Transfer data on the LIN channel.

#### Prototype

```
bool R_LIN_Transfer(
    uint8_t data1,    // Channel selection
    uint8_t data2,    // Control options
    uint8_t data3,    // ID value
    uint8_t * data4,  // Data start address
    uint8_t data5     // Data count
);
```

#### Description

Begin the transfer of data on the specified channel.

#### [data1]

Select LIN channel n (where n = 0).

#### [data2]

Control options. If multiple selections are required, use “|” to separate each selection.

- Transfer type

PDL_LIN_TX_DATA or PDL_LIN_RX_DATA	Transmit data as a response transmission or receive data as a response reception.
---------------------------------------	--

- ID parity calculation

PDL_LIN_PARITY_CALCULATE	Calculate the ID parity bits using the ID value.
--------------------------	--

- Checksum type

PDL_LIN_CHECKSUM_CLASSIC or PDL_LIN_CHECKSUM_ENHANCED	Select the classic or enhanced check sum mode.
--	--

#### [data3]

The parity and ID address to be transmitted in the header, using the format:

b7	b6	b5	b4	b3	b2	b1	b0
Parity		ID					
P1	P0	ID5	ID4	ID3	ID2	ID1	ID0

#### [data4]

The start address of the data to be sent. If no data shall be sent, specify PDL\_NO\_PTR.

#### [data5]

Set this to the number of data bytes to be transferred. The valid range is 0 to 8.

#### Return value

True if all parameters are valid and exclusive; otherwise false.

#### Category

LIN module

#### Reference

R\_LIN\_Create, R\_LIN\_GetStatus

#### Remarks

- Call R\_LIN\_Create before using this function.
- Use R\_LIN\_GetStatus to confirm that the transfer has completed.
- If the parity calculation is required, this function uses the following formulae:  
 $P0 = ID0 \oplus ID1 \oplus ID2 \oplus ID4$   
 $P1 = \neg(ID1 \oplus ID3 \oplus ID4 \oplus ID5)$

**Program example**

```
/* RPDL definitions */
#include "r_pdl_lin.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func( void )
{
    uint8_t tx_data_store[8];

    /* Send 8 bytes to ID 31h */
    R_LIN_Transfer(
        0,
        PDL_LIN_TX_DATA | \
        PDL_LIN_PARITY_CALCULATE | PDL_LIN_CHECKSUM_ENHANCED,
        0x31,
        tx_data_store,
        8
    );

    /* Receive 7 bytes from ID 15 */
    R_LIN_Transfer(
        0,
        PDL_LIN_RX_DATA | \
        PDL_LIN_PARITY_CALCULATE | PDL_LIN_CHECKSUM_CLASSIC,
        15,
        PDL_NO_PTR,
        7
    );
}
```

#### 4) R\_LIN\_Read

##### Synopsis

Store data that has been received on the LIN channel.

##### Prototype

```
bool R_LIN_Read(
    uint8_t data1,    // Channel selection
    uint8_t * data2,  // Received data start address
    uint8_t data3     // Data count
);
```

##### Description

Read the buffer data registers.

##### [data1]

Select LIN channel n (where n = 0).

##### [data2]

The start address of the storage area for the received data.

##### [data3]

Set this to the number of data bytes to be read. The valid range is from 1 to 8.

##### Return value

True if all parameters are valid and exclusive; otherwise false.

##### Category

LIN module

##### Reference

R\_LIN\_Create, R\_LIN\_GetStatus

##### Remarks

- Use R\_LIN\_GetStatus to confirm that data has been received.

##### Program example

```
/* RPDL definitions */
#include "r_pdl_lin.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func( void )
{
    uint8_t data_store[8];

    /* Read 8 bytes */
    R_LIN_Read(
        0,
        data_store,
        8
    );
}
```



## 5) R\_LIN\_Control

### Synopsis

Control the LIN channel.

### Prototype

```
bool R_LIN_Control(
    uint8_t data1,    // Channel selection
    uint8_t data2,    // Control options
    uint8_t data3     // Check sum
);
```

### Description

Modify the operation of the selected LIN channel.

#### [data1]

Select LIN channel n (where n = 0).

#### [data2]

Control the channel. All selections are optional.

If multiple selections are required, use “|” to separate each selection.

- Reset control

PDL_LIN_RESET	Enter reset mode.
---------------	-------------------

- Mode control

PDL_LIN_MODE_OPERATION or PDL_LIN_MODE_WAKE_UP or PDL_LIN_MODE_SELFTEST	Select Operation, Wake-up or Self-test mode.
---	--

- Wake-up control

PDL_LIN_WAKE_UP_TX or PDL_LIN_WAKE_UP_RX	Start a wake-up transmission or enable the detection of a wake-up transmission.
---	--

- Check sum register control (valid only in self-test mode)

PDL_LIN_CHECKSUM_WRITE	Write parameter data3 into the checksum register.
------------------------	---

#### [data3]

The check sum to be received (valid only in self-test mode; specify PDL\_NO\_DATA otherwise).

### Return value

True if all parameters are valid and exclusive; otherwise false.

### Category

LIN module

### Reference

R\_LIN\_Create, R\_LIN\_GetStatus

### Remarks

- Call R\_LIN\_Create before using this function.
- Selecting Self-test mode also selects Operation mode.
- Use Reset when changing from Self-test mode.
- Function R\_LIN\_GetStatus can be used to determine the current mode of operation.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_lin.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Select normal operation mode */
    R_LIN_Control(
        0,
        PDL_LIN_MODE_OPERATION,
        PDL_NO_DATA
    );

    /* Transmit a Wake-up signal */
    R_LIN_Control(
        0,
        PDL_LIN_MODE_WAKE_UP | PDL_LIN_WAKE_UP_TX,
        PDL_NO_DATA
    );

    /* Select self-test operation mode */
    R_LIN_Control(
        0,
        PDL_LIN_MODE_SELFTEST,
        PDL_NO_DATA
    );
}
```

## 6) R\_LIN\_GetStatus

**Synopsis**

Check the status of the LIN channel.

**Prototype**

```
bool R_LIN_GetStatus(
    uint8_t data1,    // Channel selection
    uint16_t * data2, // Status flags
    uint8_t * data3    // Checksum
);
```

**Description**

Acquires the LIN channel status.

**[data1]**

Select LIN channel n (where n = 0).

**[data2]**

The status flags shall be stored in the format below.

Specify PDL\_NO\_PTR if this information is not required

b15 – b14	b13	b12	b11	b10	b9	b8
0	Error detection					
	Check sum	-	Framing	Frame timeout	Bus	Bit
	0: No error 1: Error detected					

b7	b6	b5	b4	b3	b2 - b0
Header transmission	Data 1 reception	Error	Frame or wake-up		Mode
			Reception	Transmission	
0: No completion 1: Complete		0: None 1: Detected	0: No completion 1: Complete		xx0: Reset 001: Wake-up 011: Operation 1xx: Self-test

**[data3]**

The storage location for the check sum for the last data that was transmitted or received.

If this information is not required, specify PDL\_NO\_PTR.

**Return value**

True if all parameters are valid; otherwise false.

**Category**

LIN module

**Reference**

None.

**Remarks**

- The flags are reset when a new transfer is started using R\_LIN\_Transfer.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_lin.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    uint16_t StatusValue;

    /* Read the status of channel 0 */
    R_LIN_GetStatus(
        0,
        &StatusValue,
        PDL_NO_PTR
    );
}
```

## 4.2.21. 12-bit Analog to Digital Converter

## 1) R\_ADC\_12\_Set

**Synopsis**

Select the 12-bit ADC pins.

**Prototype**

```
bool R_ADC_12_Set(
    uint8_t data    // ADC pin selection
);
```

**Description**

Select the pin set for 12-bit ADC external triggers.

**[data]**

Select the pin set options. To set multiple options at the same time, use "|" to separate each value.

- Pin selection

PDL_ADC_12_ADTRG0_A or PDL_ADC_12_ADTRG0_B	Select the -A or -B pins for ADTRG0#.
PDL_ADC_12_ADTRG1_A or PDL_ADC_12_ADTRG1_B	Select the -A or -B pins for ADTRG1#.

**Return value**

True if all parameters are valid and exclusive; otherwise false.

**Category**

12-bit ADC

**Reference**

R\_ADC\_12\_CreateUnit

**Remarks**

- If an external trigger source is to be used and the selected device package offers A and B pins for the 12-bit ADC external trigger input, call this function before calling R\_ADC\_12\_CreateUnit.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_adc_12.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func( void )
{
    /* Select the A pin for ADTRG0# */
    R_ADC_12_Set(
        PDL_ADC_12_ADTRG0_A
    );
}
```

## 2) R\_ADC\_12\_CreateUnit

### Synopsis

Configure the 12-bit ADC unit.

### Prototype

```
bool R_ADC_12_CreateUnit(
    uint8_t data1,    // Unit selection
    uint32_t data2,    // Configuration
    uint32_t data3,    // Trigger timer selection for group 0
    uint32_t data4,    // Trigger timer selection for group 1
    uint32_t data5,    // ADC conversion clock frequency
    float data6,       // ADC input sampling time
    void * func,       // Callback function
    uint8_t data7      // Interrupt priority level
);
```

### Description (1/4)

Set the ADC mode and operating condition.

#### [data1]

Select the ADC unit (0 or 1) to be configured.

#### [data2]

Conversion options. To set multiple options at the same time, use “|” to separate each value. The default settings are shown in **bold**.

- Operation mode selection

PDL_ADC_12_MODE_SINGLE or PDL_ADC_12_MODE_ONE_CYCLE_SCAN or PDL_ADC_12_MODE_CONTINUOUS_SCAN or PDL_ADC_12_MODE_TWO_CHANNEL_SCAN	Single, Single-cycle scan, Continuous scan or 2-channel scan
--	---

- Input channel selection (n = 0 or 1)

PDL_ADC_12_CHANNELS_OPTION_0 or	Single mode: Channel ANn00.  Single-cycle scan, continuous scan or two-channel scan mode (started by software): Channel ANn00.  Two-channel scan mode (started by trigger from MTU3 / GPT or external trigger): Group 0: Channel ANn00. Group 1: Channels ANn01 to ANn03.
PDL_ADC_12_CHANNELS_OPTION_1 or	Single mode: Channel ANn01.  Single-cycle scan, continuous scan or two-channel scan mode (started by software): Channels ANn00 and ANn01.  Two-channel scan mode (started by trigger from MTU3 / GPT or external trigger): Group 0: Channels ANn00 and ANn01. Group 1: Channels ANn02 and ANn03.
PDL_ADC_12_CHANNELS_OPTION_2 or	Single mode: Channel ANn02.  Single-cycle scan, continuous scan or two-channel scan mode (started by software): Channels ANn00 to ANn02.  Two-channel scan mode (started by trigger from MTU3 / GPT or external trigger): Group 0: Channels ANn00 to ANn02. Group 1: Channel ANn03.

**Description (2/4)**

PDL_ADC_12_CHANNELS_OPTION_3	<p>Single mode: Channel ANn03.</p> <p>Single-cycle scan, continuous scan or two-channel scan mode (started by software): Channels ANn00 to ANn03.</p> <p>Two-channel scan mode (started by trigger from MTU3 / GPT or external trigger): Setting prohibited.</p>
------------------------------	--

- Trigger source enabling

PDL_ADC_12_TIMER_TRIGGER_ENABLE or PDL_ADC_12_ADTRGN_TRIGGER_ENABLE	<p>Enable ADC to be started by MTU3 or GPT. Enable ADC to be started by ADTRGn#. If neither of the options is selected, only software will be used as the trigger source.</p>
--	---

- Sample-and-hold control

PDL_ADC_12_SAMPLE_AND_HOLD_ENABLE or PDL_ADC_12_SAMPLE_AND_HOLD_DISABLE	<p>Enable or disable the sample-and-hold circuit for channels ANn00 to ANn02.</p>
--	---

- Data precision selection

PDL_ADC_12_DATA_PRECISION_12 or PDL_ADC_12_DATA_PRECISION_10 or PDL_ADC_12_DATA_PRECISION_8	<p>The accuracy of the ADC conversion result within the 16-bit register.</p>
---	--

- Result register clearing control

PDL_ADC_12_RETAIN_RESULT or PDL_ADC_12_CLEAR_RESULT	<p>Retain or clear the value in each result register after it has been read.</p>
--	--

- The alignment of the ADC conversion result within the 16-bit register.

PDL_ADC_12_DATA_ALIGNMENT_RIGHT or PDL_ADC_12_DATA_ALIGNMENT_LEFT	<p>Right: padded at the LSB end. Left: padded at the MSB end.</p>
--	---

- DTC trigger control

PDL_ADC_12_DTC_TRIGGER_DISABLE or PDL_ADC_12_DTC_TRIGGER_ENABLE	<p>Disable or enable activation of the DTC when a scan cycle completes.</p>
--	---

- 2-channel scan interrupt selection

PDL_ADC_12_EITHER_GROUP or PDL_ADC_12_BOTH_GROUP	<p>Specify interrupt generation on conversion completion of a process started by either or both of group 0 and 1 triggers.</p>
---	--

- Double trigger interrupt selection

PDL_ADC_12_EITHER_TRIGGER or PDL_ADC_12_BOTH_TRIGGER	<p>Specify interrupt generation on conversion completion of a process started by either or both of the double triggers.</p>
---	---

- Sampling time calculation

PDL_ADC_12_ADSSTR_CALCULATE or PDL_ADC_12_ADSSTR_SPECIFY	<p>Select whether parameter data6 is used to calculate the ADSSTR value, or contains the value to be stored in register ADSSTR.</p>
---	---

- Self-Diagnostic function

PDL_ADC_12_SELF_DIAGNOSTIC_DISABLE or  PDL_ADC_12_SELF_DIAGNOSTIC_VREFH0_ZERO or PDL_ADC_12_SELF_DIAGNOSTIC_VREFH0_HALF or PDL_ADC_12_SELF_DIAGNOSTIC_VREFH0_FULL or PDL_ADC_12_SELF_DIAGNOSTIC_VREFH0_ROTATED	<p>Disable the self-diagnostic function, or enable and use the voltage on pin VREFH0: x 0, x ½, x 1 or automatically rotated voltage.</p>
---	---

**Description (3/4)****[data3] and [data4]**

Single, single-cycle scan or continuous scan mode: data3 specifies the trigger timer selection, while data4 is ignored.

2-channel scan mode: data3 and data4 specify the trigger timer selections for group 0 and 1.

The trigger timer sources selected for the two groups must be different.

Specify PDL\_NO\_DATA for both, if PDL\_ADC\_12\_TIMER\_TRIGGER\_ENABLE is not selected for parameter data2.

- Trigger selection (n = 0 or 1)

PDL_ADC_12_GP_TRIGGER_MTU_0_CMIC or	Compare match or input capture A on MTU channel 0.	
PDL_ADC_12_GP_TRIGGER_MTU_1_CMIC or	Compare match or input capture A on MTU channel 1.	
PDL_ADC_12_GP_TRIGGER_MTU_2_CMIC or	Compare match or input capture A on MTU channel 2.	
PDL_ADC_12_GP_TRIGGER_MTU_3_CMIC or	Compare match or input capture A on MTU channel 3.	
PDL_ADC_12_GP_TRIGGER_MTU_4_CMIC or	Compare match or input capture A, or underflow in complementary PWM mode on MTU channel 4.	
PDL_ADC_12_GP_TRIGGER_MTU_6_CMIC or	Compare match or input capture A on MTU channel 6.	
PDL_ADC_12_GP_TRIGGER_MTU_7_CMIC or	Compare match or input capture A, or underflow in complementary PWM mode on MTU channel 7.	
PDL_ADC_12_GP_TRIGGER_MTU_0_CM_E or	Compare match E on MTU channel 0.	
PDL_ADC_12_GP_TRIGGER_MTU_4_CM_A or	Compare match with TCNT and TADCORA on MTU channel 4.	
PDL_ADC_12_GP_TRIGGER_MTU_4_CM_B or	Compare match with TCNT and TADCORB on MTU channel 4.	
PDL_ADC_12_GP_TRIGGER_MTU_4_CM_A_B or	Compare match with TNCT and TADCORA or TADCORB on MTU channel 4.	
PDL_ADC_12_GP_TRIGGER_MTU_4_CM_AB_IS or	Compare match with TNCT and TADCORA or TADCORB on MTU channel 4 (when interrupt skipping function 2 is in use).	
PDL_ADC_12_GP_TRIGGER_MTU_7_CM_A or	Compare match with TCNT and TADCORA on MTU channel 7.	
PDL_ADC_12_GP_TRIGGER_MTU_7_CM_B or	Compare match with TCNT and TADCORB on MTU channel 7.	
PDL_ADC_12_GP_TRIGGER_MTU_7_CM_A_B or	Compare match with TNCT and TADCORA or TADCORB on MTU channel 7.	
PDL_ADC_12_GP_TRIGGER_MTU_7_CM_AB_IS or	Compare match with TNCT and TADCORA or TADCORB on MTU channel 7 (when interrupt skipping function 2 is in use).	
	GPT compare match	GPT channel
PDL_ADC_12_GP_TRIGGER_GPT_0_CM_A or	GTADTRA	0
PDL_ADC_12_GP_TRIGGER_GPT_0_CM_B or	GTADTRB	0
PDL_ADC_12_GP_TRIGGER_GPT_1_CM_A or	GTADTRA	1
PDL_ADC_12_GP_TRIGGER_GPT_1_CM_B or	GTADTRB	1
PDL_ADC_12_GP_TRIGGER_GPT_2_CM_A or	GTADTRA	2
PDL_ADC_12_GP_TRIGGER_GPT_2_CM_B or	GTADTRB	2
PDL_ADC_12_GP_TRIGGER_GPT_3_CM_A or	GTADTRA	3
PDL_ADC_12_GP_TRIGGER_GPT_3_CM_B or	GTADTRB	3
PDL_ADC_12_GP_TRIGGER_GPT_0_CM_A_B or	GTADTRA or GTADTRB	0
PDL_ADC_12_GP_TRIGGER_GPT_1_CM_A_B or	GTADTRA or GTADTRB	1
PDL_ADC_12_GP_TRIGGER_GPT_2_CM_A_B or	GTADTRA or GTADTRB	2
PDL_ADC_12_GP_TRIGGER_GPT_3_CM_A_B	GTADTRA or GTADTRB	3

<b>Description (4/4)</b>	<p><b>[data5]</b> The desired frequency of the conversion clock (ADCLK) in Hertz.</p> <p><b>[data6]</b> The data to be used for the sampling state register value calculations.</p> <table border="1"> <thead> <tr> <th data-bbox="424 376 523 405"><u>Data use</u></th><th data-bbox="975 376 1139 405"><u>Parameter type</u></th></tr> </thead> <tbody> <tr> <td data-bbox="424 405 751 434">The timer period in seconds or</td><td data-bbox="975 405 1023 434">float</td></tr> <tr> <td data-bbox="424 434 847 463">The value to be put in register ADSSTR</td><td data-bbox="975 434 1046 463">uint8_t</td></tr> </tbody> </table> <p><b>[func]</b> The function to be called when the ADC conversion or scan cycle is complete. Specify PDL_NO_FUNC if no callback function is required.</p> <p><b>[data7]</b> The interrupt priority level. Select between 1 (lowest priority) and 15 (highest priority). If a callback function is specified for each ADC unit, specify the same interrupt priority level. This parameter will be ignored if PDL_NO_FUNC is specified for parameter func.</p>	<u>Data use</u>	<u>Parameter type</u>	The timer period in seconds or	float	The value to be put in register ADSSTR	uint8_t
<u>Data use</u>	<u>Parameter type</u>						
The timer period in seconds or	float						
The value to be put in register ADSSTR	uint8_t						
<b>Return value</b>	True if all parameters are valid and exclusive; otherwise false.						
<b>Category</b>	12-bit ADC						
<b>References</b>	R_CGC_Set						
<b>Remarks</b>	<ul style="list-style-type: none"> <li>• Interrupts are enabled automatically if a callback function is specified. Please see the notes on callback function usage in §6.</li> <li>• If an external trigger is used, the low-level pulse width must be at least 1.5 PCLK cycles.</li> <li>• This function brings the converter unit out of the power-down state.</li> <li>• A callback function is called by the interrupt processing function. This means that no other interrupt can be processed until the callback function returns.</li> <li>• Function R_CGC_Set must be called before any use of this function.</li> <li>• Allow 10 ms to elapse from the completion of this function to the start of the first conversion.</li> <li>• For more details of the MTU and GPT trigger options, please refer to the RX62G hardware manual.</li> <li>• For 2-channel scan mode started by External Trigger (ADTRGn#), only Group 0 will be triggered.</li> </ul>						



**Program example**

```
/* RPDL definitions */
#include "r_pdl_adc_12.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

/* ADC callback function */
void ADCIntFunc(void){}

void func(void)
{
    /* Set up the ADC for self-diagnostic with automatically
       rotated voltage value */
    R_ADC_12_CreateUnit(
        0,
        PDL_ADC_12_SELF_DIAGNOSTIC_VREFH0_ROTATED,
        PDL_NO_DATA,
        PDL_NO_DATA,
        10E6,
        20E-6,
        PDL_NO_FUNC,
        0
    );

    /* Set up the ADC in single mode using AN000 */
    R_ADC_12_CreateUnit(
        0,
        PDL_ADC_12_MODE_SINGLE | PDL_ADC_12_CHANNELS_OPTION_0 | \
        PDL_ADC_12_SAMPLE_AND_HOLD_DISABLE,
        PDL_NO_DATA,
        PDL_NO_DATA,
        10E6,
        20E-6,
        ADCIntFunc,
        7
    );
}
```

## 3) R\_ADC\_12\_CreateChannel

## Synopsis

Configure the 12-bit ADC unit.

## Prototype

```
bool R_ADC_12_CreateChannel(
    uint32_t data1, // Configuration
    uint32_t data2, // Channel 000 configuration
    uint32_t data3, // Channel 001 configuration
    uint32_t data4, // Channel 002 configuration
    uint32_t data5, // Channel 100 configuration
    uint32_t data6, // Channel 101 configuration
    uint32_t data7, // Channel 102 configuration
    void * func,    // Callback function
    uint8_t data8   // Interrupt priority level
);
```

## Description (1/2)

Set the comparator and gain amplifier.

## [data1]

Global comparator options. To set multiple options at the same time, use “|” to separate each value. The default settings are shown in **bold**.

- Comparator low reference voltage selection for AN000 to AN002

PDL_ADC_12_CMP_AN00X_REFL_EXT or PDL_ADC_12_CMP_AN00X_REFL_INT	External low reference voltage Internal low reference voltage
---	--

- Comparator low reference voltage selection for AN100 to AN102

PDL_ADC_12_CMP_AN10X_REFL_EXT or PDL_ADC_12_CMP_AN10X_REFL_INT	External low reference voltage Internal low reference voltage
---	--

- Comparator internal low reference voltage selection

<b>PDL_ADC_12_CMP_REFL_AVCC0_1_8</b> or PDL_ADC_12_CMP_REFL_AVCC0_2_8 or PDL_ADC_12_CMP_REFL_AVCC0_3_8 or PDL_ADC_12_CMP_REFL_AVCC0_4_8 or PDL_ADC_12_CMP_REFL_AVCC0_5_8 or PDL_ADC_12_CMP_REFL_AVCC0_6_8 or PDL_ADC_12_CMP_REFL_AVCC0_7_8	AVCC0 × 1/8, AVCC0 × 2/8, AVCC0 × 3/8, AVCC0 × 4/8, AVCC0 × 5/8, AVCC0 × 6/8 or AVCC0 × 7/8.
--	--

- Comparator high reference voltage selection for AN000 to AN002

PDL_ADC_12_CMP_AN00X_REFH_EXT or PDL_ADC_12_CMP_AN00X_REFH_INT	External high reference voltage Internal high reference voltage
---	--

- Comparator high reference voltage selection for AN100 to AN102

PDL_ADC_12_CMP_AN10X_REFH_EXT or PDL_ADC_12_CMP_AN10X_REFH_INT	External high reference voltage Internal high reference voltage
---	--

- Comparator internal high reference voltage selection

<b>PDL_ADC_12_CMP_REFH_AVCC0_1_8</b> or PDL_ADC_12_CMP_REFH_AVCC0_2_8 or PDL_ADC_12_CMP_REFH_AVCC0_3_8 or PDL_ADC_12_CMP_REFH_AVCC0_4_8 or PDL_ADC_12_CMP_REFH_AVCC0_5_8 or PDL_ADC_12_CMP_REFH_AVCC0_6_8 or PDL_ADC_12_CMP_REFH_AVCC0_7_8	AVCC0 × 1/8, AVCC0 × 2/8, AVCC0 × 3/8, AVCC0 × 4/8, AVCC0 × 5/8, AVCC0 × 6/8 or AVCC0 × 7/8.
--	--

- Comparator input signal selection for AN000 to AN002

<b>PDL_ADC_12_CMP_AN00X_BEFORE_AMPLIFIER</b> or PDL_ADC_12_CMP_AN00X_AFTER_AMPLIFIER	Before or after amplified by the programmable gain amplifier.
---	--

**Description (2/2)**

- Comparator input signal selection for AN100 to AN102

**PDL\_ADC\_12\_CMP\_AN10X\_BEFORE\_AMPLIFIER** or  
**PDL\_ADC\_12\_CMP\_AN10X\_AFTER\_AMPLIFIER**

Before or after amplified by the programmable gain amplifier.

- Comparator DTC trigger control

**PDL\_ADC\_12\_CMP\_DTC\_TRIGGER\_DISABLE** or  
**PDL\_ADC\_12\_CMP\_DTC\_TRIGGER\_ENABLE**

Disable or enable activation of the DTC when specified condition is detected.

- Comparator POE request control

**PDL\_ADC\_12\_CMP\_POE\_REQUEST\_DISABLE** or  
**PDL\_ADC\_12\_CMP\_POE\_REQUEST\_ENABLE**

Disable or enable POE request when specified condition is detected.

**[data2] [data3] [data4] [data5] [data6] [data7]**

Channel-specific comparator and gain amplifier options.

To set multiple options at the same time, use “|” to separate each value.

The default settings are shown in **bold**.

- Gain selection

**PDL\_ADC\_12\_CH\_GAIN\_DISABLE** or  
**PDL\_ADC\_12\_CH\_GAIN\_2\_000** or  
**PDL\_ADC\_12\_CH\_GAIN\_2\_500** or  
**PDL\_ADC\_12\_CH\_GAIN\_3\_077** or  
**PDL\_ADC\_12\_CH\_GAIN\_3\_636** or  
**PDL\_ADC\_12\_CH\_GAIN\_4\_000** or  
**PDL\_ADC\_12\_CH\_GAIN\_4\_444** or  
**PDL\_ADC\_12\_CH\_GAIN\_5\_000** or  
**PDL\_ADC\_12\_CH\_GAIN\_5\_714** or  
**PDL\_ADC\_12\_CH\_GAIN\_6\_667** or  
**PDL\_ADC\_12\_CH\_GAIN\_10\_000** or  
**PDL\_ADC\_12\_CH\_GAIN\_13\_333**

The gain factors applied to the channel.

- Comparator operating mode selection

**PDL\_ADC\_12\_CH\_CMP\_DISABLE** or  
**PDL\_ADC\_12\_CH\_CMP\_LOW** or  
**PDL\_ADC\_12\_CH\_CMP\_HIGH** or  
**PDL\_ADC\_12\_CH\_CMP\_WINDOW**

The channel is not used as the input of comparator, used as the input of low-level comparator, used as the input of high-level comparator, or used as the input of window comparator.

- Comparator operating mode selection

**PDL\_ADC\_12\_CH\_CMP\_SAMPLE\_DISABLE** or  
**PDL\_ADC\_12\_CH\_CMP\_SAMPLE\_PCLK\_DIV\_1** or  
**PDL\_ADC\_12\_CH\_CMP\_SAMPLE\_PCLK\_DIV\_2** or  
**PDL\_ADC\_12\_CH\_CMP\_SAMPLE\_PCLK\_DIV\_4** or  
**PDL\_ADC\_12\_CH\_CMP\_SAMPLE\_PCLK\_DIV\_8** or  
**PDL\_ADC\_12\_CH\_CMP\_SAMPLE\_PCLK\_DIV\_16** or  
**PDL\_ADC\_12\_CH\_CMP\_SAMPLE\_PCLK\_DIV\_128**

Comparator detection results are not sampled or sampled 16 times with  $PCLK \div 1, 2, 4, 8, 16$  or 128.

- Comparator operating mode selection

**PDL\_ADC\_12\_CH\_CMP\_INT\_POE\_DISABLE** or  
**PDL\_ADC\_12\_CH\_CMP\_INT\_POE\_ENABLE**

Disable or enable the comparator detection used as an interrupt or a POE request.

**[func]**

The function to be called when a defined event is detected by the comparator. Specify **PDL\_NO\_FUNC** if no callback function is required.

**[data 8]**

The interrupt priority level. Select between 1 (lowest priority) and 15 (highest priority). This parameter will be ignored if **PDL\_NO\_FUNC** is specified for parameter func.

**Return value**

True if all parameters are valid and exclusive; otherwise false.

**Category**

12-bit ADC

**Reference**

R\_ADC\_12\_CreateUnit

**Program example**

```
/* RPDL definitions */
#include "r_pdl_adc_12.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

/* Comparator callback function */
void CmpIntFunc(void){}

void func( void )
{
    /* Configure the channel options */
    R_ADC_12_CreateChannel(
        PDL_ADC_12_CMP_REFH_AVCC0_7_8 | PDL_ADC_12_CMP_AN00X_REFH_INT,
        PDL_ADC_12_CH_CMP_HIGH | PDL_ADC_12_CH_CMP_INT_POE_ENABLE,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        CmpIntFunc,
        7
    );
}
```

#### 4) R\_ADC\_12\_Destroy

##### Synopsis

Shut down the ADC units or comparator.

##### Prototype

```
bool R_ADC_12_Destroy(
    uint8_t data    // ADC unit and / or comparator channel selection
);
```

##### Description

Put the ADC into the Power-down state, with minimal power consumption.

##### [data]

Select the ADC units and / or comparator to be shut down.

To select multiple options at the same time, use "|" to separate each value.

- ADC unit or comparator channel selection

PDL_ADC_12_0_DESTROY	ADC unit 0
PDL_ADC_12_1_DESTROY	ADC unit 1
PDL_ADC_12_CMP_000_DESTROY	Comparator channel AN000
PDL_ADC_12_CMP_001_DESTROY	Comparator channel AN001
PDL_ADC_12_CMP_002_DESTROY	Comparator channel AN002
PDL_ADC_12_CMP_100_DESTROY	Comparator channel AN100
PDL_ADC_12_CMP_101_DESTROY	Comparator channel AN101
PDL_ADC_12_CMP_102_DESTROY	Comparator channel AN102

PDL\_ADC\_12\_CMP\_ALL\_DESTROY can be used to select all comparator channels.

##### Return value

True if a valid unit is selected; otherwise false.

##### Category

12-bit ADC

##### Reference

None.

##### Remarks

- If an ADC unit is shut down, this function implements a 1ms delay to allow the ADC to stop any current scan cycle.

##### Program example

```
/* RPDL definitions */
#include "r_pdl_adc_12.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func( void )
{
    /* Shut down ADC unit 0 */
    R_ADC_12_Destroy(
        PDL_ADC_12_0_DESTROY,
    );

    /* Shut down the comparators on channels AN001 and AN200 */
    R_ADC_12_Destroy(
        PDL_ADC_12_CMP_001_DESTROY | PDL_ADC_12_CMP_200_DESTROY
    );

    /* Shut down all the 12-bit ADC units and comparators */
    R_ADC_12_Destroy(
        PDL_ADC_12_0_DESTROY | PDL_ADC_12_0_DESTROY | \
        PDL_ADC_12_CMP_ALL_DESTROY
    );
}
```

## 5) R\_ADC\_12\_Control

### Synopsis

Start or stop an ADC unit.

### Prototype

```
bool R_ADC_12_Control(
    uint8_t data    // Conversion unit control
);
```

### Description

Start / stop operation of the specified ADC.

#### [data]

To select multiple options at the same time, use "|" to separate each value.

- On / off control for unit 0

PDL_ADC_12_0_ON or	Start a software-triggered conversion.
PDL_ADC_12_0_OFF	Stop the undergoing conversion.

- On / off control for unit 1

PDL_ADC_12_1_ON or	Start a software-triggered conversion.
PDL_ADC_12_1_OFF	Stop the undergoing conversion.

- Control the CPU during the ADC conversion. The default setting is shown in **bold**.

<b>PDL_ADC_12_CPU_ON</b> or	Allow the CPU to run normally during the conversion.
PDL_ADC_12_CPU_OFF	Stop the CPU when the conversion process starts. The CPU will re-start when any valid interrupt occurs.

### Return value

True if all parameters are valid and exclusive; otherwise false.

### Category

12-bit ADC

### Reference

R\_ADC\_12\_CreateUnit

### Remarks

- For single-cycle scan and 2-channel scan mode, the ADC will stop automatically when the conversion is complete.
- Software trigger is prevented in 2-channel scan mode with external or timer trigger enabled.

### Program example

```
/* RPDL definitions */
#include "r_pdl_adc_12.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Start the ADC conversion process */
    R_ADC_12_Control(
        PDL_ADC_12_0_ON | PDL_ADC_12_CPU_ON
    );
}
```

## 6) R\_ADC\_12\_Read

### Synopsis

Read the ADC conversion results, comparator detection flags and self-diagnostic result.

### Prototype

```
bool R_ADC_12_Read(
    uint8_t data1,      // ADC unit selection
    uint8_t * data2,    // Pointer to the buffer where comparator detection flags are to be stored
    uint16_t * data3,   // Pointer to the buffer where the converted values are to be stored
    uint16_t * data4    // Pointer to the buffer where the self-diagnostic result is to be stored
);
```

### Description

Reads the conversion results and comparator detection flags.

#### [data1]

Select the ADC unit (0 or 1) to be read.

#### [data2]

The comparator detection flags shall be stored in the format below.  
Specify PDL\_NO\_PTR if this information is not required.

b7 - b6	b5	b4	b3	b2	b1	b0
0	Specified condition detection (0 = Not detected, 1 = Detected)					
	AN102	AN101	AN100	AN002	AN001	AN000

#### [data3]

Specify a pointer to a variable or an array where the results shall be stored.

Specify PDL\_NO\_PTR if this information is not required.

From 1 to 5 two-byte memory slots are required.

The number depends on the settings for A/D conversion mode and "Input channel selection" when R\_ADC\_12\_CreateUnit is used to configure the ADC unit.

Except for conversion result from ANn00 in the case below, all the conversion results are stored in the ascending order of channel number.

When an MTU3 or GPT trigger is selected, two memory slots are required for channel ANn00 (n = 0 or 1), if being active. One occupies the first place in the array and the other occupies the last. If the conversion is triggered by "TRGnAN (n = 4 or 7) in the MTU3" or "GTADTRAnN (n = 0 to 3) in the GPT", the first memory slot contains the result from channel ANn00 and the last one contains undefined data; if the conversion is triggered by "TRGnBN (n = 4 or 7) in the MTU3" or "GTADTRBnN (n = 0 to 3) in the GPT", the first memory slot contains undefined and the last one contains the result from channel ANn00.

The data alignment and accuracy are controlled using the R\_ADC\_12\_CreateUnit function.

For 2-channel scan mode started by External Trigger (ADTRGn#), the user must prepare buffer area for both Group 0 and Group 1. However, only Group 0 will produce valid conversion results.

#### [data4]

Specify a pointer to a variable where the self-diagnostic result shall be stored.

Specify PDL\_NO\_PTR if this information is not required.

Refer to hardware manual Section 27.2.1 (2) for the format of self-diagnostic result.

### Return value

True if a valid unit is selected; otherwise false.

### Category

12-bit ADC

### Reference

R\_ADC\_12\_CreateUnit

**Remarks**

- Ensure that the storage area is big enough for the requested number of results.
- If no callback function is used, this function waits for the ADST flag to indicate that conversion is complete before reading the results. If the ADC unit's control registers are directly modified by the user, this function may lock up.
- Examples of data3 format:

Two channel scan mode.

AN000 selected as group 0, triggered by GTADTRB0N;

AN001, AN002, AN003 selected as group 1, triggered by TRGA2N.

data3[0]	Invalid	ADDR0A
data3[1]	Vaild	ADDR1
data3[2]	Vaild	ADDR2
data3[3]	Vaild	ADDR3
data3[4]	Vaild	ADDR0B

Single cycle scan mode.

AN000 and AN001 selected; triggered by TRG7AN.

data3[0]	Vaild	ADDR0A
data3[1]	Vaild	ADDR1
data3[2]	Invalid	ADDR0B

Single cycle scan mode.

AN000 and AN001 selected; triggered by software.

data3[0]	Vaild	ADDR0A
data3[1]	Vaild	ADDR1

**Program example**

```

/* RPD_L definitions */
#include "r_pdl_adc_12.h"

/* RPD_L device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    uint8_t CMPflags;
    /* 2 slots for AN000 and 3 slots for all the other channels */
    uint16_t ADCresult[5];
    uint16_t Diagresult;
    /* Read the ADC */
    R_ADC_12_Read(
        0,
        &CMPflags,
        ADCresult,
        &Diagresult
    );
}

```



## 4.2.22. 10-bit Analog to Digital Converter

## 1) R\_ADC\_10\_Create

**Synopsis**

Configure an ADC unit.

**Prototype**

```
bool R_ADC_10_Create(
    uint8_t data1, // ADC unit selection
    uint32_t data2, // ADC configuration
    uint32_t data3, // ADC trigger selection
    uint32_t data4, // ADC conversion clock frequency
    float data5,   // ADC input sampling time
    void * func,   // Callback function
    uint8_t data6  // Interrupt priority level
);
```

**Description (1/4)**

Set the ADC's mode and operating condition.

**[data1]**

Select the ADC unit (0 only) to be configured.

**[data2]**

Conversion options. To set multiple options at the same time, use “|” to separate each value. The default settings are shown in **bold**.

- Input channel selection

PDL_ADC_10_CHANNELS_OPTION_0 or	Any mode: Channel AN0.
PDL_ADC_10_CHANNELS_OPTION_1 or	Single mode: Channel AN1. Scan mode: Channels AN0 to AN1.
PDL_ADC_10_CHANNELS_OPTION_2 or	Single mode: Channel AN2. Scan mode: Channels AN0 to AN2.
PDL_ADC_10_CHANNELS_OPTION_3 or	Single mode: Channel AN3. Scan mode: Channels AN0 to AN3.
PDL_ADC_10_CHANNELS_OPTION_4 or	Single mode: Channel AN4. Scan mode: Channels AN0 to AN4.
PDL_ADC_10_CHANNELS_OPTION_5 or	Single mode: Channel AN5. Scan mode: Channels AN0 to AN5.
PDL_ADC_10_CHANNELS_OPTION_6 or	Single mode: Channel AN6. Scan mode: Channels AN0 to AN6.
PDL_ADC_10_CHANNELS_OPTION_7 or	Single mode: Channel AN7. Scan mode: Channels AN0 to AN7.
PDL_ADC_10_CHANNELS_OPTION_8 or	Single mode: Channel AN8. Scan mode: Channels AN0 to AN8.
PDL_ADC_10_CHANNELS_OPTION_9 or	Single mode: Channel AN9. Scan mode: Channels AN0 to AN9.
PDL_ADC_10_CHANNELS_OPTION_10 or	Single mode: Channel AN10. Scan mode: Channels AN0 to AN10.
PDL_ADC_10_CHANNELS_OPTION_11	Single mode: Channel AN11. Scan mode: Channels AN0 to AN11.

**Description (2/4)**

## • Operation mode

PDL\_ADC\_10\_MODE\_SINGLE or  
PDL\_ADC\_10\_MODE\_CONTINUOUS\_SCAN or  
PDL\_ADC\_10\_MODE\_ONE\_CYCLE\_SCAN

Select single mode, continuous scan mode or one-cycle scan mode.

## • Data accuracy

PDL\_ADC\_10\_DATA\_ACCURACY\_10\_BIT or  
PDL\_ADC\_10\_DATA\_ACCURACY\_8\_BIT

The accuracy of the ADC conversion result within the 16-bit register.

## • Data alignment selection

PDL\_ADC\_10\_DATA\_ALIGNMENT\_RIGHT or  
PDL\_ADC\_10\_DATA\_ALIGNMENT\_LEFT

The alignment of the ADC conversion result within the 16-bit register.  
Left: padded at the MSB end.  
Right: padded at the LSB end.

## • DTC trigger control

PDL\_ADC\_10\_DTC\_TRIGGER\_DISABLE or  
PDL\_ADC\_10\_DTC\_TRIGGER\_ENABLE

Disable or enable activation of the DTC when a conversion or scan cycle completes.

## • Sampling time calculation

PDL\_ADC\_10\_ADSSTR\_CALCULATE or  
PDL\_ADC\_10\_ADSSTR\_SPECIFY

Select whether parameter data5 is used to calculate the ADSSTR value, or contains the value to be stored in register ADSSTR.

## • Self-Diagnostic function

PDL\_ADC\_10\_SELF\_DIAGNOSTIC\_DISABLE or

PDL\_ADC\_10\_SELF\_DIAGNOSTIC\_VREF\_ZERO or  
PDL\_ADC\_10\_SELF\_DIAGNOSTIC\_VREF\_HALF or  
PDL\_ADC\_10\_SELF\_DIAGNOSTIC\_VREF\_FULL

Disable the self-diagnostic function, or enable and use the voltage on pin VREF  
x 0,  
x 1/2, or  
x 1.

**Description (3/4)****[data3]**

Select the ADC trigger source.

PDL_ADC_10_TRIGGER_SOFTWARE or	Software trigger.
PDL_ADC_10_TRIGGER_ADTRG or	ADTRG# pin.
PDL_ADC_10_TRIGGER_MTU_0_CMIC or	Compare match or input capture A on MTU channel 0.
PDL_ADC_10_TRIGGER_MTU_1_CMIC or	Compare match or input capture A on MTU channel 1.
PDL_ADC_10_TRIGGER_MTU_2_CMIC or	Compare match or input capture A on MTU channel 2.
PDL_ADC_10_TRIGGER_MTU_3_CMIC or	Compare match or input capture A on MTU channel 3.
PDL_ADC_10_TRIGGER_MTU_4_CMIC or	Compare match or input capture A, or underflow in complementary PWM mode on MTU channel 4.
PDL_ADC_10_TRIGGER_MTU_6_CMIC or	Compare match or input capture A on MTU channel 6.
PDL_ADC_10_TRIGGER_MTU_7_CMIC or	Compare match or input capture A, or underflow in complementary PWM mode on MTU channel 7.
PDL_ADC_10_TRIGGER_MTU_0_CM_E or	Compare match E on MTU channel 0.
PDL_ADC_10_TRIGGER_MTU_4_CM_A or	Compare match with TCNT and TADCORA on MTU channel 4.
PDL_ADC_10_TRIGGER_MTU_4_CM_B or	Compare match with TCNT and TADCORB on MTU channel 4.
PDL_ADC_10_TRIGGER_MTU_4_CM_AB or	Compare match with TNCT and TADCORA or TADCORB on MTU channel 4.
PDL_ADC_10_TRIGGER_MTU_4_CM_AB_IS or	Compare match with TNCT and TADCORA or TADCORB on MTU channel 4 (when interrupt skipping function 2 is in use).
PDL_ADC_10_TRIGGER_MTU_7_CM_A or	Compare match with TCNT and TADCORA on MTU channel 7.
PDL_ADC_10_TRIGGER_MTU_7_CM_B or	Compare match with TCNT and TADCORB on MTU channel 7.
PDL_ADC_10_TRIGGER_MTU_7_CM_AB or	Compare match with TNCT and TADCORA or TADCORB on MTU channel 7.
PDL_ADC_10_TRIGGER_MTU_7_CM_AB_IS or	Compare match with TNCT and TADCORA or TADCORB on MTU channel 7 (when interrupt skipping function 2 is in use).
PDL_ADC_10_TRIGGER_GPT_0_CM_A or	Compare match GTADTRA on GPT channel 0.
PDL_ADC_10_TRIGGER_GPT_0_CM_B or	Compare match GTADTRB on GPT channel 0.
PDL_ADC_10_TRIGGER_GPT_1_CM_A or	Compare match GTADTRA on GPT channel 1.
PDL_ADC_10_TRIGGER_GPT_1_CM_B or	Compare match GTADTRB on GPT channel 1.
PDL_ADC_10_TRIGGER_GPT_2_CM_A or	Compare match GTADTRA on GPT channel 2.
PDL_ADC_10_TRIGGER_GPT_2_CM_B or	Compare match GTADTRB on GPT channel 2.
PDL_ADC_10_TRIGGER_GPT_3_CM_A or	Compare match GTADTRA on GPT channel 3.
PDL_ADC_10_TRIGGER_GPT_3_CM_B or	Compare match GTADTRB on GPT channel 3.
PDL_ADC_10_TRIGGER_GPT_0_CM_AB or	Compare match GTADTRA or GTADTRB on GPT channel 0.
PDL_ADC_10_TRIGGER_GPT_1_CM_AB or	Compare match GTADTRA or GTADTRB on GPT channel 1.
PDL_ADC_10_TRIGGER_GPT_2_CM_AB or	Compare match GTADTRA or GTADTRB on GPT channel 2.
PDL_ADC_10_TRIGGER_GPT_3_CM_AB	Compare match GTADTRA or GTADTRB on GPT channel 3.

**[data4]**

The desired frequency of the conversion clock (ADCLK) in Hertz.

<b>Description (4/4)</b>	<p><b>[data5]</b> The data to be used for the sampling state register value calculations.</p> <table> <tr> <td><u>Data use</u></td><td><u>Parameter type</u></td></tr> <tr> <td>The timer period in seconds or</td><td>float</td></tr> <tr> <td>The value to be put in register ADSSTR</td><td>uint8_t</td></tr> </table> <p><b>[func]</b> The function to be called when the ADC conversion or scan cycle is complete. Specify PDL_NO_FUNC if no callback function is required.</p> <p><b>[data6]</b> The interrupt priority level. Select between 1 (lowest priority) and 15 (highest priority). This parameter will be ignored if PDL_NO_FUNC is specified for parameter func.</p>	<u>Data use</u>	<u>Parameter type</u>	The timer period in seconds or	float	The value to be put in register ADSSTR	uint8_t
<u>Data use</u>	<u>Parameter type</u>						
The timer period in seconds or	float						
The value to be put in register ADSSTR	uint8_t						
<b>Return value</b>	True if all parameters are valid and exclusive; otherwise false.						
<b>Category</b>	ADC						
<b>References</b>	R_CGC_Set						
<b>Remarks</b>	<ul style="list-style-type: none"> <li>This function configures the selected pin(s) for ADC operation by setting the direction to input and turning off the input buffer. The port control settings for any ADC pins that subsequently become inactive are not modified.</li> <li>This function brings the converter unit out of the power-down state.</li> <li>Interrupts are enabled automatically if a callback function is specified. Please see the notes on callback function usage in §6.</li> <li>A callback function is executed by the interrupt processing function. This means that no other interrupt can be processed until the callback function has completed.</li> <li>Function R_CGC_Set must be called before any use of this function.</li> <li>The available values for the conversion clock are <math>PCLK \div 8, 4, 2</math> or <math>1</math>. If the desired frequency is not an exact match, the actual frequency will be the next highest frequency. The timing limits depend on the peripheral module clock, PCLK.</li> <li>The 10-bit ADC is not available on the 64-pin package.</li> </ul>						

Parameter	Limit	Equation	$f_{PCLK}$ (MHz)			
			50	12.5	32	8
Conversion clock (ADCLK)	Minimum	$(f_{PCLK} \div 8, 4, 2, 1) \geq 4.0$	6.25 MHz	6.25 MHz	4.0 MHz	4.0 MHz
	Maximum	$f_{PCLK}$	50.00 MHz	12.50 MHz	32.00 MHz	8.00 MHz
Conversion time	Maximum	$25 \div ADCLK$	4.0 $\mu$ s	4.0 $\mu$ s	6.25 $\mu$ s	6.25 $\mu$ s
	Minimum		0.5 $\mu$ s	2.0 $\mu$ s	0.781 $\mu$ s	3.13 $\mu$ s
Sampling time	Minimum	-	0.5 $\mu$ s			
	Maximum	$255 \div ADCLK$	e.g. 5.1 $\mu$ s at 50 MHz			
<b>Total conversion time</b>	Minimum	Conversion time + sampling time	1.0 $\mu$ s	2.5 $\mu$ s	1.28 $\mu$ s	3.63 $\mu$ s

**Program example**

```
/* RPDL definitions */
#include "r_pdl_adc_10.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

/* ADC unit callback function */
void ADCIntFunc(void){}

void func(void)
{
    /* Set up ADC at 50 MHz in single mode using AN1 with 0.6μs sampling
    time */
    R_ADC_10_Create(
        0,
        PDL_ADC_10_MODE_SINGLE | PDL_ADC_10_CHANNELS_OPTION_1,
        PDL_ADC_10_TRIGGER_ADTRG,
        50E6,
        0.6E-6,
        ADCIntFunc,
        2
    );
}
```

## 2) R\_ADC\_10\_Destroy

### Synopsis

Shut down an ADC unit.

### Prototype

```
bool R_ADC_10_Destroy(
    uint8_t data // ADC unit selection
);
```

### Description

Put the ADC into the Power-down state, with minimal power consumption.

#### [data]

Select the ADC unit (0 only) to be shut down.

### Return value

True if a valid unit is selected; otherwise false.

### Category

ADC

### Reference

R\_ADC\_10\_Create

### Remarks

- This function waits for the ADST flag to indicate that the converter has stopped. If the ADC unit's control registers are directly modified by the user, this function may lock up.

### Program example

```
/* RPD_L definitions */
#include "r_pdl_adc_10.h"

/* RPD_L device-specific definitions */
#include "r_pdl_definitions.h"

void func( void )
{
    /* Shut down ADC */
    R_ADC_10_Destroy(
        0
    );
}
```

### 3) R\_ADC\_10\_Control

#### Synopsis

Start or stop an ADC unit.

#### Prototype

```
bool R_ADC_10_Control(
    uint8_t data    // Conversion unit control
);
```

#### Description

Controls start / stop operation of the specified ADC.

#### [data]

To select multiple options at the same time, use "|" to separate each value.

- On / off control

PDL_ADC_10_0_ON or PDL_ADC_10_0_OFF	Start or stop ADC conversion.
--	-------------------------------

- Control the CPU during the ADC conversion. The default setting is shown in **bold**.

<b>PDL_ADC_10_CPU_ON</b> or	Allow the CPU to run normally during the conversion.
PDL_ADC_10_CPU_OFF	Stop the CPU when the conversion starts. The CPU will re-start when any valid interrupt occurs.

#### Return value

True if all parameters are valid and exclusive; otherwise false.

#### Category

ADC

#### Reference

R\_ADC\_10\_Create

#### Remarks

- Use this API function only when the software trigger option is selected.
- For single or one-cycle scan modes, the ADC will stop automatically when the conversion is complete.

#### Program example

```
/* RPDL definitions */
#include "r_pdl_adc_10.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    /* Stop ADC unit */
    R_ADC_10_Control(
        PDL_ADC_10_0_OFF
    );
}
```

## 4) R\_ADC\_10\_Read

**Synopsis**

Read the ADC conversion results.

**Prototype**

```
bool R_ADC_10_Read(
    uint8_t data1,    // ADC unit selection
    uint16_t * data2  // Pointer to the buffer where the converted values are to be stored
);
```

**Description**

Read the conversion values for the ADC unit.

**[data1]**

Select the ADC unit (0 only) to be read.

**[data2]**

Specify a pointer to a variable or array where the results shall be stored.

**Return value**

True if a valid unit is selected; otherwise false.

**Category**

ADC

**Reference**

R\_ADC\_10\_Create

**Remarks**

- Up to 12 conversion results will be read and stored. The number depends on the settings for "Input channel selection" and "Scan mode" when R\_ADC\_10\_Create is used to configure the ADC unit.
- The data alignment and accuracy is controlled using the R\_ADC\_10\_Create function.
- Ensure that the buffer is big enough for the requested number of values.
- If no callback function is used, this function waits for the ADI flag, indicating that conversion is complete, before reading the results. If the ADC unit's control registers are directly modified by the user, this function may lock up.

**Program example**

```
/* RPDL definitions */
#include "r_pdl_adc_10.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void func(void)
{
    uint16_t ADCresult[2];

    /* Read the ADC values */
    R_ADC_10_Read(
        0,
        ADCresult
    );
}
```



## 5. Usage Examples

This chapter shows programming examples for each driver in this library.

### 5.1. Interrupt control

Figure 5-1 shows an example of external interrupt use.

Pin IRQ0-A is used to detect a falling edge and generates an interrupt.

Pin IRQ1-B is used to detect a falling edge and is polled.

Pin IRQ2-A is used to detect a low-level signal and generates an interrupt. Further interrupts are prevented until the signal has returned to the high level.

```
/* Peripheral driver function prototypes */
#include "r_pdl_intc.h"
#include "r_pdl_io_port.h"

/* RPD_L device-specific definitions */
#include "r_pdl_definitions.h"

volatile uint8_t switch_sw1_pressed;
volatile uint8_t irq2_low;

/* Callback function prototypes */
void IRQ0Handler(void);
void IRQ0Handler(void);
static void ReEnableIRQ2(void);

void main(void)
{
    uint8_t irq_status;

    /* Set the CPU's Interrupt Priority Level to 0 */
    R_INTC_Write(
        PDL_INTC_REG_IPL,
        PDL_NO_DATA,
        0
    );

    /* Configure the IRQ0 interrupt on pin IRQ0-A */
    R_INTC_CreateExtInterrupt(
        PDL_INTC_IRQ0,
        PDL_INTC_FALLING | PDL_INTC_A,
        IRQ0Handler,
        7
    );

    /* Configure the IRQ1 interrupt on pin IRQ1-B */
    R_INTC_CreateExtInterrupt(
        PDL_INTC_IRQ1,
        PDL_INTC_FALLING | PDL_INTC_B,
        PDL_NO_FUNC,
        0
    );

    /* Configure the IRQ2 interrupt on pin IRQ2-A */
    R_INTC_CreateExtInterrupt(
        PDL_INTC_IRQ2,
        PDL_INTC_LOW | PDL_INTC_A,
        IRQ2Handler,
        7
    );

    irq2_low = false;
}
```

```

while(1)
{
    /* Poll the IRQ1 flag */
    R_INTC_GetExtInterruptStatus(
        PDL_INTC_IRQ1,
        &irq_status
    );
    if ((irq_status & 0x01) == 0)
    {
        /* Disable IRQ1 */
        R_INTC_ControlExtInterrupt(
            PDL_INTC_IRQ1,
            PDL_INT_DISABLE
        );
    }

    /* Has IRQ2 triggered? */
    if (irq2_low == true)
    {
        /* Re-enable the interrupt if the signal has returned to the high level */
        R_IO_PORT_Compare(
            PDL_IO_PORT_3_2,
            1,
            ReEnableIRQ2
        );
    }
}

void IRQ0Handler(void)
{
    /* Process the IRQ0 event here (the flag is cleared automatically) */
    switch_sw1_pressed = true;
}

void IRQ2Handler(void)
{
    /* Disable the level-triggered interrupt */
    R_INTC_ControlExtInterrupt(
        PDL_INTC_IRQ2,
        PDL_INTC_DISABLE
    );
    irq2_low = true;
}

static void ReEnableIRQ2(void)
{
    /* Re-enable the interrupt (and try to clear it) */
    R_INTC_ControlExtInterrupt(
        PDL_INTC_IRQ2,
        PDL_INTC_ENABLE | PDL_INTC_CLEAR_IR_FLAG
    );
}

```

Figure 5-1: Example of External Interrupt

## 5.2. I/O Port

Figure 5-2 shows examples of I/O port configuration, reading and writing.

```

/* Peripheral driver function prototypes */
#include "r_pdl_io_port.h"
#include "r_pdl_pfc.h"

/* RPD_L device-specific definitions */
#include "r_pdl_definitions.h"

void main(void)
{
    uint8_t result;

    /* Configure port 4 as an input */
    R_IO_PORT_Set(
        PDL_IO_PORT_4,
        PDL_IO_PORT_INPUT | PDL_IO_PORT_INPUT_BUFFER_ON
    );

    /* Configure port pin P21 as an output */
    R_IO_PORT_Set(
        PDL_IO_PORT_2_1,
        PDL_IO_PORT_OUTPUT
    );

    /* Write 0x44 to register PFC2 */
    R_PFC_Write(
        2,
        0x44
    );

    /* Read the value of all the pins on port 4 */
    R_IO_PORT_Read(
        PDL_IO_PORT_4,
        &result
    );

    /* Set pin P21 to output high */
    R_IO_PORT_Write(
        PDL_IO_PORT_2_1,
        1
    );

    /* Invert pin P21 */
    R_IO_PORT_Modify(
        PDL_IO_PORT_2_1,
        PDL_IO_PORT_XOR,
        1
    );

    /* And the value on port 4 with 55h */
    R_IO_PORT_Modify(
        PDL_IO_PORT_4,
        PDL_IO_PORT_AND,
        0x55
    );

    /* Read the control registers for port PC */
    R_IO_PORT_ReadControl(
        PDL_IO_PORT_1,
        &direction,
        &buffer,
        &pull_up,
        &output
    );
}

```

```
/* Read the direction for pin P03 */
R_IO_PORT_ReadControl(
    PDL_IO_PORT_0_3,
    &direction,
    PDL_NO_PTR,
    PDL_NO_PTR,
    PDL_NO_PTR
);

/* Set the lower 4 bits on port P1 to output */
R_IO_PORT_ModifyControl(
    PDL_IO_PORT_1,
    PDL_IO_PORT_DIRECTION | PDL_IO_PORT_OR,
    0x0F
);

/* Enable the pull-up on pin PA3 */
R_IO_PORT_ModifyControl(
    PDL_IO_PORT_A_3,
    PDL_IO_PORT_PULL_UP | PDL_IO_PORT_OR,
    1
);
}
```

**Figure 5-2: Example of I/O Port Operations**

### 5.3. Voltage Detection Circuit

Figure 5-3 shows an example of Voltage detection circuit usage.

If the supply voltage drops below 3.15V (Vdet2), the callback function is called.

If the supply voltage drops below 2.85V (Vdet1), the MCU is reset.

```
/* Peripheral driver function prototypes */
#include "r_pdl_lvd.h"
#include "r_pdl_intc.h"

/* PDL device-specific definitions */
#include "r_pdl_definitions.h"

void NMICallBackFunc(void);

void main(void)
{
    /* Generate an NMI when Vdet2 is reached; reset if Vdet1 is reached */
    R_LVD_Control(
        PDL_LVD_VDET2_INTERRUPT_VDET1_RESET
    );

    /* Configure the NMI */
    R_INTC_CreateExtInterrupt(
        PDL_INTC_NMI,
        PDL_INTC_FALLING | PDL_INTC_LVD_ENABLE,
        NMICallBackFunc,
        PDL_NO_DATA
    );
}

void NMICallBackFunc(void)
{
    /* Handle the low voltage detection response here */
}
```

**Figure 5-3: Example of Independent Watchdog Timer use**

## 5.4. Bus Controller

Figure 5-4 shows an example of external bus controller usage.

```
/* Peripheral driver function prototypes */
#include "r_pdl_bsc.h"
#include "r_pdl_cgc.h"
#include "r_pdl_intc.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void BSC_error_handler(void)
{
    /* Clear the error signals */
    R_BSC_Control(
        PDL_BSC_ERROR_CLEAR
    );
}

void main(void)
{
    /* Set the CPU's Interrupt Priority Level to 0 */
    R_INTC_Write(
        PDL_INTC_REG_IPL,
        0
    );

    /* Configure the bus controller */
    R_BSC_Create(
        PDL_BSC_ERROR_ILLEGAL_ADDRESS_ENABLE,
        BSC_error_handler,
        5
    );

    /* Initialise the system clocks */
    R_CGC_Set(
        12.5E6,
        100E6,
        50E6,
        PDL_NO_DATA
    );
}
```

Figure 5-4: Example of Bus Controller use

## 5.5. Data Transfer Controller

### 5.5.1. Block transfer mode

Figure 5-5 shows an example of Data Transfer Controller usage.

```

/* Peripheral driver function prototypes */
#include "r_pdl_dtc.h"
#include "r_pdl_intc.h"

/* RPDLC device-specific definitions */
#include "r_pdl_definitions.h"

/* Required for this example */
#include <string.h>

/* Reserve an area for the DTC vector table */
#pragma address dtc_vector_table = 0x00001000
uint32_t dtc_vector_table[256];

/* Reserve 16 bytes (full address mode) for the transfer data areas */
uint32_t dtc_irq0_transfer_data[4];
uint32_t dtc_sw_transfer_data[4];

const char source_string_1[]="Renesas RX62G";
const char source_string_2[]="DTC example";

volatile char destination_string_1[]=".....";
volatile char destination_string_2[]=".....";
volatile uint8_t transfer_complete;

void main(void)
{
    uint16_t StatusValue;
    uint32_t SourceAddr;
    uint32_t DestAddr;
    uint16_t Count;
    uint8_t Block_Size;

    /* Configure the software interrupt */
    R_INTC_CreateSoftwareInterrupt(
        PDL_INTC_DTC_SW_TRIGGER_ENABLE,
        PDL_NO_FUNC,
        0
    );

    /* Enable the SW1 interrupt */
    R_INTC_CreateExtInterrupt(
        PDL_INTC_IRQ0,
        PDL_INTC_FALLING | PDL_INTC_B | PDL_INTC_DTC_TRIGGER_ENABLE,
        PDL_NO_FUNC,
        0
    );

    /* Configure the controller */
    R_DTC_Set(
        PDL_DTC_ADDRESS_FULL,
        dtc_vector_table
    );

    /* Configure the DTC for IRQ0 */
    R_DTC_Create(
        PDL_DTC_BLOCK | PDL_DTC_SOURCE | \
        PDL_DTC_SOURCE_ADDRESS_PLUS | PDL_DTC_DESTINATION_ADDRESS_PLUS | \
        PDL_DTC_SIZE_8 | PDL_DTC_TRIGGER_IRQ0,
        dtc_irq0_transfer_data,
        source_string_1,

```

```

        destination_string_1,
        1,
        (uint8_t)strlen(source_string_1)
    );

    /* Configure the DTC for Software trigger */
    R_DTC_Create(
        PDL_DTC_BLOCK | PDL_DTC_SOURCE | \
        PDL_DTC_SOURCE_ADDRESS_PLUS | PDL_DTC_DESTINATION_ADDRESS_PLUS | \
        PDL_DTC_SIZE_8 | PDL_DTC_TRIGGER_SW,
        dtc_sw_transfer_data,
        source_string_2,
        destination_string_2,
        1,
        (uint8_t)strlen(source_string_2)
    );

    /* Start the controller */
    R_DTC_Control(
        PDL_DTC_START,
        PDL_NO_PTR,
        PDL_NO_PTR,
        PDL_NO_PTR,
        PDL_NO_DATA,
        PDL_NO_DATA
    );

    /* Read the status and current source address for the Software transfer */
    R_DTC_GetStatus(
        dtc_sw_transfer_data,
        &StatusValue,
        &SourceAddr,
        &DestAddr,
        &Count,
        &Block_Size
    );

    /* Generate a software interrupt request */
    R_INTC_Write(
        PDL_INTC_REG_SWINTR,
        1
    );

    /* Read the status and current source address for the Software transfer */
    R_DTC_GetStatus(
        dtc_sw_transfer_data,
        &StatusValue,
        &SourceAddr,
        &DestAddr,
        &Count,
        &Block_Size
    );

do
{
    /* Read the status and current source address for the IRQ0 transfer */
    R_DTC_GetStatus(
        dtc_irq0_transfer_data,
        &StatusValue,
        &SourceAddr,
        &DestAddr,
        &Count,
        &Block_Size
    );
} while (Count != 0);

/* Shutdown the DTC */

```

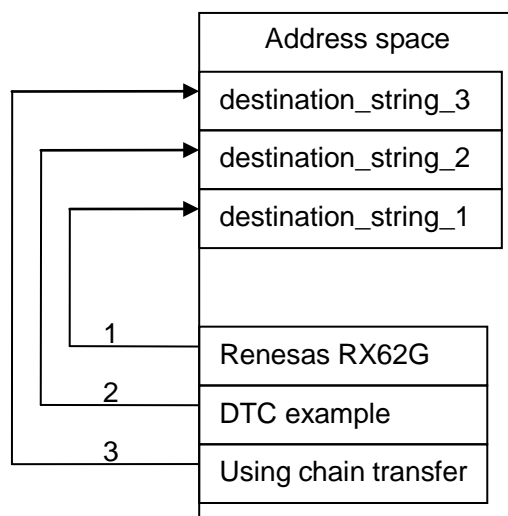


```
R_DTC_Destroy(  
    );  
  
while(1);  
}
```

**Figure 5-5: Example of DTC use**

## 5.5.2. Chain transfer operation

Figure 5-6 shows an example of Data Transfer Controller operation, using chain transfer of blocks.



Transfer 1 is triggered by a software interrupt and copies data from ROM into RAM.  
 On completion of transfer 1, transfer 2 is started.  
 On completion of transfer 2, transfer 3 is started.

```

/* Peripheral driver function prototypes */
#include "r_pdl_dtc.h"
#include "r_pdl_intc.h"

/* RPDLC device-specific definitions */
#include "r_pdl_definitions.h"

/* Required for this example */
#include <string.h>

/* Reserve an area for the DTC vector table */
#pragma address dtc_vector_table = 0x00001000
uint32_t dtc_vector_table[256];

/* Reserve three contiguous groups of 16 bytes (full address mode) for the transfer data
areas */
uint32_t dtc_sw_transfer_data[4 * 3];

const char source_string_1[]="Renesas RX62G";
const char source_string_2[]="DTC example";
const char source_string_3[]="using chain transfer";

volatile char destination_string_1[]=".....";
volatile char destination_string_2[]=".....";
volatile char destination_string_3[]=".....";
volatile uint8_t transfer_complete;

void main(void)
{
    uint16_t StatusValue;
    uint32_t SourceAddr;
    uint32_t DestAddr;
    uint16_t Count;
    uint8_t Block_Size;

```

```

/* Enable software interrupts */
R_INTC_CreateSoftwareInterrupt(
    PDL_INTC_DTC_SW_TRIGGER_ENABLE,
    PDL_NO_FUNC,
    0
);

/* Configure the controller */
R_DTC_Set(
    PDL_DTC_ADDRESS_FULL,
    dtc_vector_table
);

/* Configure the DTC for Software trigger */
R_DTC_Create(
    PDL_DTC_BLOCK | PDL_DTC_SOURCE | \
    PDL_DTC_SOURCE_ADDRESS_PLUS | PDL_DTC_DESTINATION_ADDRESS_PLUS | \
    PDL_DTC_SIZE_8 | PDL_DTC_CHAIN_0 | PDL_DTC_TRIGGER_SW,
    dtc_sw_transfer_data,
    source_string_1,
    destination_string_1,
    1,
    (uint8_t)strlen(source_string_1)
);

/* Configure the DTC for chain transfer */
R_DTC_Create(
    PDL_DTC_BLOCK | PDL_DTC_SOURCE | \
    PDL_DTC_SOURCE_ADDRESS_PLUS | PDL_DTC_DESTINATION_ADDRESS_PLUS | \
    PDL_DTC_SIZE_8 | PDL_DTC_CHAIN_0 | PDL_DTC_TRIGGER_CHAIN,
    dtc_sw_transfer_data + 4,
    source_string_2,
    destination_string_2,
    1,
    (uint8_t)strlen(source_string_2)
);

/* Configure the DTC for chain transfer */
R_DTC_Create(
    PDL_DTC_BLOCK | PDL_DTC_SOURCE | \
    PDL_DTC_SOURCE_ADDRESS_PLUS | PDL_DTC_DESTINATION_ADDRESS_PLUS | \
    PDL_DTC_SIZE_8 | PDL_DTC_TRIGGER_CHAIN,
    dtc_sw_transfer_data + 8,
    source_string_3,
    destination_string_3,
    1,
    (uint8_t)strlen(source_string_3)
);

/* Start the controller */
R_DTC_Control(
    PDL_DTC_START,
    PDL_NO_PTR,
    PDL_NO_PTR,
    PDL_NO_PTR,
    PDL_NO_DATA,
    PDL_NO_DATA
);

/* Read the status and current source address for the Software transfer */
R_DTC_GetStatus(
    dtc_sw_transfer_data,
    &StatusValue,
    &SourceAddr,
    &DestAddr,
    &Count,
    &Block_Size
);

```

```
/* Generate a software interrupt request */
R_INTC_Write(
    PDL_INTC_REG_SWINTR,
    1
);

/* Read the status and current source address for the Software transfer */
R_DTC_GetStatus(
    dtc_sw_transfer_data,
    &StatusValue,
    &SourceAddr,
    &DestAddr,
    &Count,
    &Block_Size
);

/* Shutdown the DTC */
R_DTC_Destroy(
);

while(1);
}
```

Figure 5-6: Example of DTC chain transfer

## 5.6. General PWM Timer Driver

Figure 5-7 shows a usage example of General PWM Timer Driver.

```

/* Peripheral driver function prototypes */
#include "r_pdl_gpt.h"
#include "r_pdl_cgc.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void main(void)
{
    R_GPT_ControlChannel_structure gpt_ch_control_parameters;
    R_GPT_Create_structure ch_create_parameters;

    /* Initialise the system clocks */
    NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.

    /* Select the GPT pins */
    R_GPT_Set(
        PDL_GPT_PINS_AB_A
    );

    /* Load the defaults */
    R_GPT_Create_load_defaults(&ch_create_parameters);

    /* Set the non-default options */
    ch_create_parameters.data2 = PDL_GPT_MODE_SAW | PDL_GPT_CLK_ICLK_DIV_1;
    ch_create_parameters.data6 = PDL_GPT_A_CM_INVERT | PDL_GPT_A_LOW_LOW | \
        PDL_GPT_A_CYCLE_RETAIN;
    ch_create_parameters.data7 = PDL_GPT_B_CM_INVERT | PDL_GPT_B_LOW_LOW | \
        PDL_GPT_B_CYCLE_RETAIN;

    /* Configure channel 0 for dual-waveform (A and B) output */
    R_GPT_Create(
        0,
        &ch_create_parameters
    );

    /* Set the register values for channel 0 */
    gpt_ch_control_parameters.data4 = 0x0000;
    gpt_ch_control_parameters.data5 = 0x2222;
    gpt_ch_control_parameters.data6 = 0xBBBB;
    gpt_ch_control_parameters.data11 = 0xFFDD;

    /* Load the register values and start the channel */
    R_GPT_ControlChannel(
        3,
        PDL_GPT_START | PDL_GPT_COUNT_DIRECTION_UP,
        PDL_GPT_REGISTER_CYCLE | PDL_GPT_REGISTER_COUNTER | \
        PDL_GPT_REGISTER_A | PDL_GPT_REGISTER_B,
        &gpt_ch_control_parameters
    );
}

```

**Figure 5-7: Example of General PWM Timer Driver**

### 5.7. Compare Match Timer

Figure 5-8 shows an example of Compare Match Timer usage. One channel is used to generate interrupts at regular intervals.

```

/* Peripheral driver function prototypes */
#include "r_pdl_cmt.h"
#include "r_pdl_cgc.h"
#include "r_pdl_io_port.h"
#include "r_pdl_intc.h"

/* RPD_L device-specific definitions */
#include "r_pdl_definitions.h"

/* Callback function prototype */
void CMT0_handler(void);

void main(void)
{
    /* Initialise the system clocks */
    R_CGC_Set(
        12.5E6,
        100E6,
        50E6,
        PDL_NO_DATA
    );

    /* Set the CPU's Interrupt Priority Level to 0 */
    R_INTC_Write(
        PDL_INTC_REG_IPL,
        0
    );

    /* Configure a port pin for output */
    R_IO_PORT_Set(
        PDL_IO_PORT_7_1,
        PDL_IO_PORT_OUTPUT
    );

    /* Configure CMT channel 0 for 1kHz operation */
    R_CMT_Create(
        0,
        PDL_CMT_FREQUENCY,
        1E3,
        CMT0_handler,
        7
    );

    /* Change the frequency to 10kHz */
    R_CMT_Control(
        0,
        PDL_CMT_FREQUENCY,
        10E3
    );
}

void CMT0_handler(void)
{
    /* Invert the port pin */
    R_IO_PORT_Modify(
        PDL_IO_PORT_7_1,
        PDL_IO_PORT_XOR,
        1
    );
}

```

**Figure 5-8: Example of Compare Match Timer use**

## 5.8. Independent Watchdog Timer

Figure 5-9 shows an example of Independent Watchdog timer usage.

At start-up the underflow is checked to identify if the the reset was caused by the Independent Watchdog timer. The watchdog timer is then configured for a 1024-count timeout period and started.

Because the watchdog timer is not refreshed, after two seconds (this depends on the frequency of the on-chip oscillator) the MCU is reset and the underflow condition is detected.

```
/* Peripheral driver function prototypes */
#include "r_pdl_iwdt.h"

/* PDL device-specific definitions */
#include "r_pdl_definitions.h"

void main(void)
{
    uint16_t Status;

    /* Read the timer status */
    R_IWDT_Read(
        &Status
    );

    /* Has an underflow occurred? */
    if ((Status & BIT_14) != 0x0u)
    {
        /* Handle the watchdog-induced reset here */
    }

    /* Configure the IWDT */
    R_IWDT_Set(
        PDL_IWDT_TIMEOUT_1024 | PDL_IWDT_CLOCK_OCO_256
    );

    /* Start the IWDT */
    R_IWDT_Control(
        PDL_IWDT_REFRESH
    );
}
```

**Figure 5-9: Example of Independent Watchdog Timer use**

## 5.9. Serial Communication Interface

### 5.9.1. SCI Reception

Figure 5-10 shows the setting of SCI channels 0 and 1 and the reception of data using interrupts (channel 0) and polling (channel 1).

```

/* Peripheral driver function prototypes */
#include "r_pdl_sci.h"
#include "r_pdl_cgc.h"
#include "r_pdl_intc.h"

/* RPD_L device-specific definitions */
#include "r_pdl_definitions.h"

/* Callback function prototypes */
void System_failed(void){};
void System_reset(void){};
void SCI0RxFunc(void);
void SCI0ErrFunc(uint8_t);

volatile char rx_string[10];

void main(void)
{
    uint8_t result;

    /* Initialise the system clocks */
    R_CGC_Set(
        12.5E6,
        100E6,
        50E6,
        PDL_NO_DATA
    );

    /* Set the CPU's Interrupt Priority Level to 0 */
    R_INTC_Write(
        PDL_INTC_REG_IPL,
        PDL_NO_DATA,
        0
    );

    /* Set up SCI channel 0: Async, 8N1, 38400 baud */
    R_SCI_Create(
        0,
        PDL_SCI_ASYNC | PDL_SCI_8N1,
        38400,
        1
    );

    /* Set up SCI channel 1: Async, 8N1, 19200 baud */
    R_SCI_Create(
        1,
        PDL_SCI_ASYNC | PDL_SCI_8N1,
        19200,
        0
    );

    /* Start the interrupt-based reception of 9 characters on channel 0 */
    R_SCI_Receive(
        0,
        PDL_NO_DATA,
        rx_string,
        9,
        SCI0RxFunc,
        SCI0ErrFunc
    );
}

```



```

    /* Wait for 1 character to be received on channel 1 */
    R_SCI_Receive(
        1,
        PDL_NO_DATA,
        &result,
        1,
        PDL_NO_FUNC,
        PDL_NO_FUNC
    );

    /* Check that channel 0 has completed */
    do
    {
        R_SCI_GetStatus(
            0,
            &result,
            PDL_NO_PTR,
            PDL_NO_PTR
        );
    } while ((result & 0x10) != 0);

    /* Shut down channel 0 */
    R_SCI_Destroy(
        0
    );
}

/* SCI channel 0 receive data handler */
void SCI0RxFunc(void)
{
    char * str_ptr = rx_string;
    while (*str_ptr-- != 0)
    {
        /* Process the string contents */
    }
}

/* SCI channel 0 error handler */
void SCI0ErrFunc(void)
{
    uint8_t error_flags;

    /* Read the status */
    R_SCI_GetStatus(
        0,
        &error_flags,
        PDL_NO_PTR,
        PDL_NO_PTR
    );

    /* Overrun error? */
    if ((error_flags & 0x20) != 0)
    {
        System_failed();
    }
}

```

Figure 5-10: Example of SCI Reception code

## 5.9.2. SCI Transmission

Figure 5-11 shows the configuration of SCI channels 0 and 1 and the transmission of data (on channel 0) using polling and then interrupts.

```

/* Peripheral driver function prototypes */
#include "r_pdl_sci.h"
#include "r_pdl_cgc.h"
#include "r_pdl_intc.h"

/* RPD_L device-specific definitions */
#include "r_pdl_definitions.h"

/* Callback function prototype */
void SCI0TxFunc(void);

volatile char result[2];

void main(void)
{
    /* Put a null at the end */
    result[1] = 0;

    /* Initialise the system clocks */
    R_CGC_Set(
        12.5E6,
        100E6,
        50E6,
        PDL_NO_DATA
    );

    /* Set the CPU's Interrupt Priority Level to 0 */
    R_INTC_Write(
        PDL_INTC_REG_IPL,
        PDL_NO_DATA,
        0
    );

    /* Set up SCI channel 0: Async, 8N1, 38400 baud */
    R_SCI_Create(
        0,
        PDL_SCI_ASYNC | PDL_SCI_8N1,
        38400,
        0
    );

    /* Set up SCI channel 1: Async, 8N1, 19200 baud */
    R_SCI_Create(
        1,
        PDL_SCI_ASYNC | PDL_SCI_8N1,
        19200,
        0
    );

    /* Wait for 1 character to be received on channel 1 */
    R_SCI_Receive(
        1,
        PDL_NO_DATA,
        result,
        1,
        PDL_NO_FUNC,
        PDL_NO_FUNC
    );

    /* Send a string on channel 0 (wait for completion) */
    R_SCI_Send(
        0,
        PDL_NO_DATA,

```

```
    "Renesas RX",
    0,
    PDL_NO_FUNC
);

/* Send another string on channel 0 */
R_SCI_Send(
    0,
    PDL_NO_DATA,
    "www.renesas.com",
    0,
    SCI0TxFunc
);

/* Echo the character on channel 1 */
R_SCI_Send(
    1,
    PDL_NO_DATA,
    result,
    0,
    PDL_NO_FUNC
);
}

/* SCI channel 0 transmit complete handler */
void SCI0TxFunc(void)
{
    /* Shut down channel 0 */
    R_SCI_Destroy(
        0
    );
}
```

**Figure 5-11: Example of SCI Transmission code**

## 5.9.3. Synchronous Transmission and Reception

Figure 5-12 shows the configuration of SCI channel 2 as the clock master followed by the simultaneous transmission and reception of data.

The Receive function call uses interrupts while the Transmit function uses polling.

```

/* Peripheral driver function prototypes */
#include "r_pdl_sci.h"
#include "r_pdl_cgc.h"
#include "r_pdl_intc.h"

/* RPD_L device-specific definitions */
#include "r_pdl_definitions.h"

/* Callback function prototype */
void SCI2RxFunc(void);

volatile uint8_t data_received;

#define BUFFER_SIZE 10

void main(void)
{
    uint8_t Tx_Data[BUFFER_SIZE];
    uint8_t Rx_Data[BUFFER_SIZE];
    uint8_t transfer_size = 8;
    uint8_t i;

    /* Configure the system clocks */
    R_CGC_Set(
        12.5E6,
        100E6,
        50E6,
        PDL_NO_DATA
    );

    /* Set the CPU's Interrupt Priority Level to 0 */
    R_INTC_Write(
        PDL_INTC_REG_IPL,
        PDL_NO_DATA,
        0
    );

    /* Set up SCI channel 2: Sync, MSb first, 2 Mbps */
    R_SCI_Create(
        2,
        PDL_SCI_SYNC | PDL_SCI_MSB_FIRST,
        2E6,
        1
    );

    /* Load the required data into the transmit buffer */
    for (i = 0; i < BUFFER_SIZE; i++)
    {
        Tx_Data[i] = (uint8_t)(i + 1);
    }

    data_received = false;

    /* Set up the receive process (no bus activity will occur) */
    R_SCI_Receive(
        2,
        PDL_NO_DATA,
        Rx_Data,
        transfer_size,
        SCI2RxFunc,
        PDL_NO_FUNC
    );
}

```

```
);

/* Send data (which will also receive data at the same time) */
R_SCI_Send(
    2,
    PDL_NO_DATA,
    Tx_Data,
    transfer_size,
    PDL_NO_FUNC
);

/* Ensure the receive interrupt has processed the last byte */
while (data_received == false);

/* Process the received data here */
}

/* SCI channel 2 receive complete handler */
void SCI2RxFunc (void)
{
    data_received = true;
}
```

**Figure 5-12: Example of Synchronous Transmission and Reception code**

## 5.9.4. SCI Reception in Asynchronous Multi-Processor mode

Figure 5-13 shows the setting of SCI channel 2 and the Multi-Processor mode reception of data using interrupts and polling.

```

/* PDL functions */
#include "r_pdl_sci.h"
#include "r_pdl_cgc.h"
#include "r_pdl_io_port.h"

/* PDL device-specific definitions */
#include "r_pdl_definitions.h"

void SCIlrx(void);
void SCIlEr(void);

#define NUM_DATA 50
volatile uint8_t data_received;
volatile uint8_t error_happen;
volatile uint8_t receive_data[NUM_DATA];

void main(void)
{
    uint8_t i;
    bool id_received;

    for (i=0; i<NUM_DATA; i++)
    {
        receive_data[i] = 0;
    }

    /* Configure the clocks */
    R_CGC_Set(
        12.5E6,
        100E6,
        50E6,
        PDL_NO_DATA
    );

    /* Configure the pin selection of SCI */
    R_SCI_Set(PDL_SCI_PIN_SCI2_A);

    /* Configure the RS232 port, specify Async MP mode */
    R_SCI_Create(
        2,
        PDL_SCI_8N1 | PDL_SCI_ASYNC_MP,
        100E3,
        15
    );

    /* ----- */
    /* Async MP mode, data Reception, by CPU ISR */
    /* ----- */

    data_received = false;
    error_happen = false;

    /* Wait using interrupts, until receive matching Station ID (0x0A) */
    R_SCI_Receive(
        2,
        0x0A00 | PDL_SCI_MP_ID_CYCLE,
        PDL_NO_PTR,
        0,
        SCIlrx,
        SCIlEr
    );
};

```

```

while (data_received == false);

data_received = false;

// Receive data (ID = 0x0A) by CPU ISR
R_SCI_Receive(
    2,
    PDL_NO_DATA,
    receive_data,
    10,
    SCI1rx,
    SCI1Er
);

while (data_received == false);

/* ----- */
/* Async MP mode, data Reception, by polling */
/* ----- */
id_received = false;

/* Wait by polling, until receive matching Station ID (0x01) */
id_received = R_SCI_Receive(
    2,
    0x0100 | PDL_SCI_MP_ID_CYCLE,
    PDL_NO_PTR,
    0,
    PDL_NO_FUNC,
    SCI1Er
);

if (id_received == true)
{
    /* Receive data (ID = 0x01) by polling */
    R_SCI_Receive(
        2,
        PDL_NO_DATA,
        receive_data,
        10,
        PDL_NO_FUNC,
        SCI1Er
    );
}

}

void SCI1rx(void)
{
    data_received = true;
}

void SCI1Er(void)
{
    error_happen = true;
}

```

**Figure 5-13: Example of SCI Reception code in Asynchronous Multi-Processor mode**

## 5.9.5. SCI Transmission in Asynchronous Multi-Processor mode

Figure 5-14 shows the setting of SCI channel 2 and the Multi-Processor mode transmission of data using interrupts and polling.

```

/* PDL functions */
#include "r_pdl_sci.h"
#include "r_pdl_cgc.h"

/* PDL device-specific definitions */
#include "r_pdl_definitions.h"

void SCItx(void);

uint8_t* send_data0 = "\n\rWelcome to the Renesas RX62G.\n\r";
uint8_t* send_data = "testing ASYNC MP mode";
bool tx_end;

void main(void)
{
    /* Configure the clocks */
    R_CGC_Set(
        12.5E6,
        100E6,
        50E6,
        PDL_NO_DATA
    );

    /* Configure the pin selection of SCI */
    R_SCI_Set(PDL_SCI_PIN_SCI2_A);

    /* Configure the RS232 port, specify Async MP mode */
    R_SCI_Create(
        2,
        PDL_SCI_8N1 | PDL_SCI_ASYNC_MP,
        100E3,
        15
    );

    /* ----- */
    /* Async MP mode, data Transmission, by CPU ISR */
    /* ----- */
    /* Send Target Station ID (0x0A), using polling */
    R_SCI_Send(
        2,
        0x0A00 | PDL_SCI_MP_ID_CYCLE,
        PDL_NO_PTR,
        0,
        PDL_NO_FUNC
    );

    tx_end = false;

    /* Send data to Target Station (ID = 0x0A), using interrupts */
    R_SCI_Send(
        2,
        PDL_NO_DATA,
        send_data0,
        0,
        SCItx
    );

    while(tx_end == false);

    /* ----- */
    /* Async MP mode, data Transmission, by polling */
    /* ----- */
}

```



```
/* Send Target Station ID (0x01) by internal polling */
R_SCI_Send(
    2,
    0x0100 | PDL_SCI_MP_ID_CYCLE,
    PDL_NO_PTR,
    0,
    PDL_NO_FUNC
);

/* Send data to Target Station (ID = 0x01), by polling */
R_SCI_Send(
    2,
    PDL_NO_DATA,
    send_data,
    0,
    PDL_NO_FUNC
);
}

void SCI1tx(void)
{
    tx_end = true;
}
```

**Figure 5-14: Example of SCI Transmission code in Asynchronous Multi-Processor mode**

### 5.10. CRC calculator

Figure 5-15 shows an example of CRC usage.

The payload and CRC checksum have been received from a remote unit.

The CRC calculator is used to check that the payload is correct.

```
/* Peripheral driver function prototypes */
#include "r_pdl_crc.h"

/* RPDLC device-specific definitions */
#include "r_pdl_definitions.h"

void main(void)
{
    uint16_t crc_result;

    /* Configure the CRC to use the CCITT polynomial; LSB first */
    R_CRC_Create(
        PDL_CRC_POLY_CRC_CCITT | PDL_CRC_LSB_FIRST
    );

    /* Write the payload data */
    R_CRC_Write(
        0xF0
    );

    /* Write the first half of the CRC checksum */
    R_CRC_Write(
        0x8F
    );

    /* Write the second half of the CRC checksum */
    R_CRC_Write(
        0xF7
    );

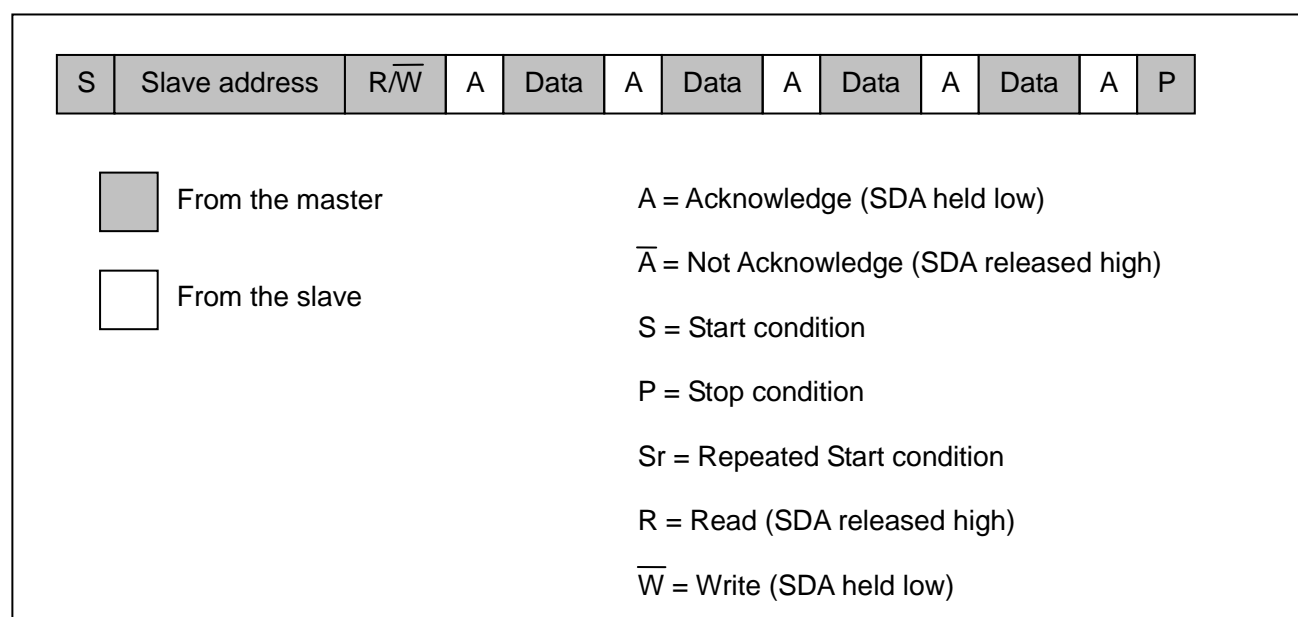
    /* Read the CRC calculation result */
    R_CRC_Read(
        PDL_NO_DATA,
        & crc_result
    );

    /* Shutdown the CRC unit */
    R_CRC_Destroy(
    );
}
```

Figure 5-15: Example of CRC calculation

### 5.11. I<sup>2</sup>C Bus Interface

In the following examples, the bus activity will be illustrated using the following format.



**Figure 5-16: I<sup>2</sup>C bus activity notation**

#### 5.11.1. Master mode

In this example an EEPROM device has been connected to channel 0.

The EEPROM responds to the 7-bit slave address 1010xxxb.

During a read process the bits “xxx” can be any value.

During a write process:

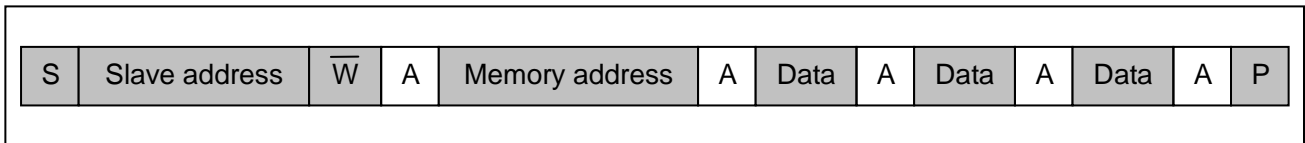
- i) The bits “xxx” represent the EEPROM memory address bits a10, a9 and a8.
- ii) The first byte after the slave address is the EEPROM memory address bits a7 to a0.

The EEPROM has a write cycle time of 5 ms.

The following examples illustrate the use of Master mode.

## 1) Configuration and transmission

The MCU's I<sup>2</sup>C channel 1 will be configured for Master operation and used to send three bytes to a slave.



**Figure 5-17: The bus activity, showing 4 bytes being transmitted to the EEPROM**

```

/* Peripheral driver function prototypes */
#include "r_pdl_iic.h"
#include "r_pdl_cgc.h"
#include "r_pdl_intc.h"
#include "r_pdl_cmt.h"

/* RPD_L device-specific definitions */
#include "r_pdl_definitions.h"

#define EEPROM_ADDRESS 0xA0

void main(void)
{
    const uint8_t eeprom_data_array_1[5] = {0x00, 0x01, 0x02, 0x03, 0x04};
    uint16_t status_flags = 0;
    uint16_t TxChars;
    uint16_t RxChars;

    /* Initialise the system clocks */
    R_CGC_Set(
        12.5E6,
        100E6,
        50E6,
        PDL_NO_DATA
    );

    /* Set the CPU's Interrupt Priority Level to 0 */
    R_INTC_Write(
        PDL_INTC_REG_IPL,
        PDL_NO_DATA,
        0
    );

    /* Select I2C mode at 100kHz, 300ns rise time, 200ns fall time */
    R_IIC_Create(
        0,
        PDL_IIC_MODE_IIC | PDL_IIC_INT_PCLK_DIV_8,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        100E3,
        (300 << 16) | 200
    );

    /* Send the lower address and 3 bytes to the EEPROM, using polling */
    if (R_IIC_MasterSend(
        0,
        PDL_NO_DATA,
        EEPROM_ADDRESS,
        eeprom_data_array_1,
        4,
        PDL_NO_FUNC,
        0
    )

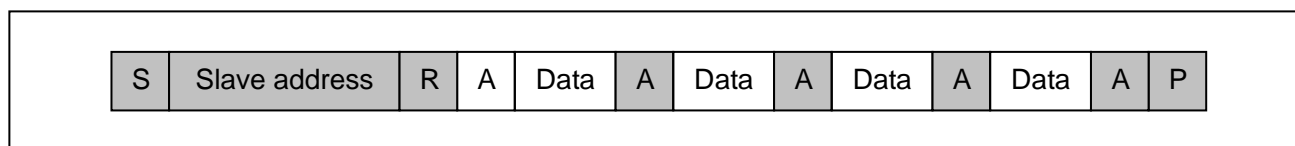
```

```
) == false)
{
    /* Read the channel and transfer status */
    R_IIC_GetStatus(
        0,
        &status_flags,
        &TxChars,
        PDL_NO_PTR
    );
    /* Review the flags and transmit count to decide on the next action */
}
else
{
    /* Wait for 5ms while the EEPROM updates */
    R_CMT_CreateOneShot(
        0,
        0,
        5E-3,
        PDL_NO_FUNC,
        0
    );
}
```

**Figure 5-18: Configure the I<sup>2</sup>C channel and write 3 data bytes to the first locations**

## 2) Reception

I<sup>2</sup>C channel 1 will be configured for Master operation and used to read four bytes from a slave device.



**Figure 5-19: The bus activity, showing 4 bytes being transmitted by the EEPROM**

```

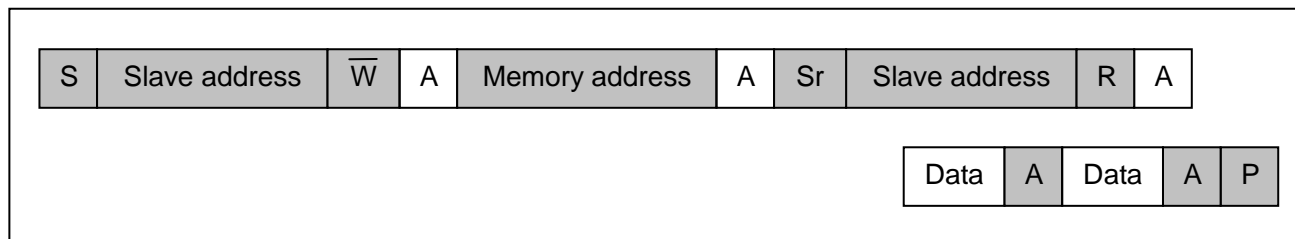
/* Read data from the EEPROM, using polling */
if (R_IIC_MasterReceive(
    0,
    PDL_NO_DATA,
    EEPROM_ADDRESS,
    data_storage,
    4,
    PDL_NO_FUNC,
    0
) == false)
{
    /* Read the channel and transfer status */
    R_IIC_GetStatus(
        0,
        &status_flags,
        PDL_NO_PTR,
        &RxChars
    );
    /* Review the flags and transmit count to decide on the next action */
}

```

**Figure 5-20: An example of reading data from the EEPROM**

## 3) Repeated Start

I<sup>2</sup>C channel 1 will be configured for Master operation. The memory address pointer of an EEPROM will be modified, and then a Repeat Start condition used to change to read the byte at that memory location in the EEPROM.



**Figure 5-21: The bus activity, showing the Repeated Start condition when switching to the Read process**

```

/* Send 1 byte to the EEPROM to update the lower address bits and do not stop */
R_IIC_MasterSend(
    0,
    PDL_IIC_STOP_DISABLE,
    EEPROM_ADDRESS,
    0x37,
    1,
    PDL_NO_FUNC,
    0
);

/* Read data from the EEPROM. A repeated start will occur. */
R_IIC_MasterReceive(
    0,
    PDL_NO_DATA,
    EEPROM_ADDRESS,
    data_storage,
    2,
    PDL_NO_FUNC,
    0
);

```

**Figure 5-22: Set the lower address to 37h and then read 2 bytes.**

## 5.11.2. Master mode with DTC

In the following example, data is written to an EEPROM in two bursts. The DTC is used to handle the data transfer.

The same EEPROM address locations are then read out in two bursts. The DTC is used to handle the data transfer.

```

/* Peripheral driver function prototypes */
#include "r_pdl_iic.h"
#include "r_pdl_cgc.h"
#include "r_pdl_cmt.h"
#include "r_pdl_dtc.h"

/* RPD_L device-specific definitions */
#include "r_pdl_definitions.h"

static void write_eeprom_data(void);
static void read_eeprom_data(void);
void iic_tx_dtc_end_handler(void);
void iic_rx_dtc_end_handler(void);

#define EEPROM_MEMORY_ADDRESS_UPPER 0x00
#define EEPROM_MEMORY_ADDRESS_LOWER 0x00
#define EEPROM_ADDRESS (0x00A0 | EEPROM_MEMORY_ADDRESS_UPPER)

volatile uint8_t bus_busy;
volatile uint8_t data_storage[20];

/* Reserve an area for the DTC vector table */
#pragma address dtc_vector_table = 0x00001000
uint32_t dtc_vector_table[256];

/* Reserve 16 bytes (full address mode) for the transfer data areas */
uint32_t dtc_iic1_tx_transfer_data[4];
uint32_t dtc_iic1_rx_transfer_data[4];

void main(void)
{
    const uint8_t eeprom_data_array_1[] = {EEPROM_MEMORY_ADDRESS_LOWER, 0x01, 0x02, 0x03,
    0x04, 0x05};
    const uint8_t eeprom_data_array_2[] = {EEPROM_MEMORY_ADDRESS_LOWER + 5, 0x06, 0x07,
    0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F};
    uint8_t i;

    /* Configure the clocks */
    R_CGC_Set(
        12.5E6,
        100E6,
        50E6,
        PDL_NO_DATA
    );

    /* Configure the DTC controller */
    R_DTC_Set(
        PDL_DTC_ADDRESS_FULL,
        dtc_vector_table
    );

    /* Set up a DTC channel for IIC transmission */
    R_DTC_Create(
        PDL_DTC_NORMAL | \
        PDL_DTC_SOURCE_ADDRESS_PLUS | PDL_DTC_DESTINATION_ADDRESS_FIXED | \
        PDL_DTC_SIZE_8 | PDL_DTC_IRQ_COMPLETE | PDL_DTC_TRIGGER_ICTXI1,
        dtc_iic1_tx_transfer_data,
        eeprom_data_array_1,
        (uint8_t *)&RIIC1.ICDRT,
        6,

```



```

    PDL_NO_DATA
);

/* Set up a DTC channel for IIC reception */
R_DTC_Create(
    PDL_DTC_NORMAL | \
    PDL_DTC_SOURCE_ADDRESS_FIXED | PDL_DTC_DESTINATION_ADDRESS_PLUS | \
    PDL_DTC_SIZE_8 | PDL_DTC_IRQ_COMPLETE | PDL_DTC_TRIGGER_ICRX11,
    dtc_iic1_rx_transfer_data,
    (uint8_t *)&RIIC1.ICDRR,
    data_storage,
    4,
    PDL_NO_DATA
);

/* Select I2C mode at 100kHz, 300ns rise time, 200ns fall time */
R_IIC_Create(
    0,
    PDL_IIC_MODE_IIC | PDL_IIC_INT_PCLK_DIV_8,
    0,
    0,
    0,
    0,
    100E3,
    (300 << 16) | 200
);

/* Enable the DTC */
R_DTC_Control(
    PDL_DTC_START,
    PDL_NO_PTR,
    PDL_NO_PTR,
    PDL_NO_PTR,
    PDL_NO_DATA,
    PDL_NO_DATA
);

/* Write the data into the EEPROM */
write_eeprom_data();

/* Prepare the next data for the EEPROM */
R_DTC_Control(
    PDL_DTC_UPDATE_SOURCE | PDL_DTC_UPDATE_COUNT,
    dtc_iic1_tx_transfer_data,
    eeprom_data_array_2,
    PDL_NO_PTR,
    8,
    PDL_NO_DATA
);

/* Write the data into the EEPROM */
write_eeprom_data();

/* Clear the data storage area */
for (i = 0; i < 20; i++) data_storage[i] = 0x00;

/* Reset the EEPROM sub-address to 0, using polling */
R_IIC_MasterSend(
    0,
    PDL_IIC_STOP_DISABLE,
    EEPROM_ADDRESS,
    eeprom_data_array_1,
    1,
    PDL_NO_FUNC,
    0
);

/* Read data from the EEPROM on channel 1, using the DTC */

```

```

read_eeeprom_data();

/* Prepare the next data */
R_DTC_Control(
    PDL_DTC_UPDATE_DESTINATION | PDL_DTC_UPDATE_COUNT,
    dtc_iic1_rx_transfer_data,
    PDL_NO_PTR,
    &data_storage[5],
    5,
    PDL_NO_DATA
);

/* Read data from the EEPROM on channel 1, using the DTC */
read_eeeprom_data();
}

static void write_eeeprom_data(void)
{
    bus_busy = true;
    /* Send data to the EEPROM on channel 1, using the DTC */
    R_IIC_MasterSend(
        0,
        PDL_IIC_DTC_TRIGGER_ENABLE,
        EEPROM_ADDRESS,
        PDL_NO_PTR,
        0,
        PDL_NO_FUNC,
        0
    );
    while (bus_busy == true);

    /* Wait for 5ms while the EEPROM updates */
    R_CMT_CreateOneShot(
        0,
        0,
        5E-3,
        PDL_NO_FUNC,
        0
    );
}

static void read_eeeprom_data(void)
{
    bus_busy = true;
    /* Read data from the EEPROM on channel 1, using the DTC */
    R_IIC_MasterReceive(
        0,
        PDL_IIC_DTC_TRIGGER_ENABLE,
        EEPROM_ADDRESS,
        PDL_NO_PTR,
        0,
        PDL_NO_FUNC,
        0
    );
    while (bus_busy == true);
}

void iic_tx_dtc_end_handler(void)
{
    uint32_t status_flags = 0;

    /* Wait for the transmission to complete */
    do
    {
        R_IIC_GetStatus(
            0,
            &status_flags,
            PDL_NO_PTR,

```

```

        PDL_NO_PTR
    );
} while((status_flags & 0x00000080ul) == 0x0u);

/* Issue a Stop condition on channel 1 */
R_IIC_Control(
    0,
    PDL_IIC_STOP
);

bus_busy = false;
}

void iic_rx_dtc_end_handler(void)
{
    uint32_t DestAddr = 0;

    /* Read the next destination address for the current transfer */
    R_DTC_GetStatus(
        dtc_iic1_rx_transfer_data,
        PDL_NO_PTR,
        PDL_NO_PTR,
        &DestAddr,
        PDL_NO_PTR,
        PDL_NO_PTR
    );

    /* Read one more byte with NACK condition on channel 1 and stop */
    R_IIC_MasterReceiveLast(
        0,
        (uint8_t *)DestAddr
    );

    bus_busy = false;
}

```

**Figure 5-23: An example of write data to and reading data from an EEPROM, using two DTC channels**

## 5.11.3. Slave mode

In this example the MCU behaves as a slave device on channel 0.  
It will respond to 7-bit address 0001001b or 10-bit address 0010010010b.

```

/* Peripheral driver function prototypes */
#include "r_pdl_iic.h"
#include "r_pdl_cgc.h"
#include "r_pdl_intc.h"
#include "r_pdl_cmt.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

void slave_event_handler(void);

const uint8_t mcu_data_array[10] = {0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80,
0x5A};
volatile bool all_data_read;
volatile bool all_data_sent;
volatile uint8_t slave_data_storage[10];

void main(void)
{
    /* Initialise the system clocks */
    R_CGC_Set(
        12.5E6,
        100E6,
        50E6,
        PDL_NO_DATA
    );

    /* Set the CPU's Interrupt Priority Level to 0 */
    R_INTC_Write(
        PDL_INTC_REG_IPL,
        PDL_NO_DATA,
        0
    );

    /* Select I2C mode at 100kHz, 300ns rise time, 200ns fall time */
    R_IIC_Create(
        0,
        PDL_IIC_MODE_IIC | PDL_IIC_INT_PCLK_DIV_8,
        PDL_IIC_SLAVE_0_ENABLE_7 | PDL_IIC_SLAVE_1_ENABLE_10,
        0x12,
        0x0124,
        PDL_NO_DATA,
        100E3,
        (300 << 16) | 200
    );

    all_data_read = false;

    /* Monitor the channel. Any data received will be stored in the receive buffer */
    R_IIC_SlaveMonitor(
        0,
        PDL_NO_DATA,
        slave_data_storage,
        10,
        slave_event_handler,
        7
    );

    while (all_data_read == false);
}

void slave_event_handler(void)
{

```

```

uint16_t status_flags = 0;
uint16_t tx_count = 0;
uint16_t rx_count = 0;

/* Read the channel status */
R_IIC_GetStatus(
    0,
    &status_flags,
    &tx_count,
    &rx_count
);

/* Slave address 0 detected with a Read? */
if ((status_flags & 0x0047) == 0x0041)
{
    /* Assign 5 bytes to be read by a master */
    R_IIC_SlaveSend(
        0,
        PDL_NO_DATA,
        mcu_data_array,
        5
    );
}

/* Slave address 1 detected with a Read? */
else if ((status_flags & 0x0047) == 0x0042)
{
    /* Assign 5 bytes to be read by a master */
    R_IIC_SlaveSend(
        0,
        PDL_NO_DATA,
        slave_data_storage,
        5
    );
}

/* NACK and Stop detected */
else if ((status_flags & 0x1800) == 0x1800)
{
    all_data_sent = true;
}

/* Stop detected */
else if ((status_flags & 0x1800) == 0x0800)
{
    all_data_read = true;
    do
    {
        R_IIC_GetStatus(
            0,
            &status_flags,
            PDL_NO_PTR,
            PDL_NO_PTR
        );
    } while ((status_flags & 0x8000) != 0x0u);
}
else
{
    /* Process any other conditions here */
}
}

```

**Figure 5-24: Configure the I<sup>2</sup>C channel and write 3 data bytes to the first locations**

## 5.12. Serial Peripheral Interface

### 5.12.1. Master operation with multiple slaves

This is an example of Serial Peripheral Interface usage where one SPI master communicates with four SPI slaves. Each slave requires different data bit lengths.

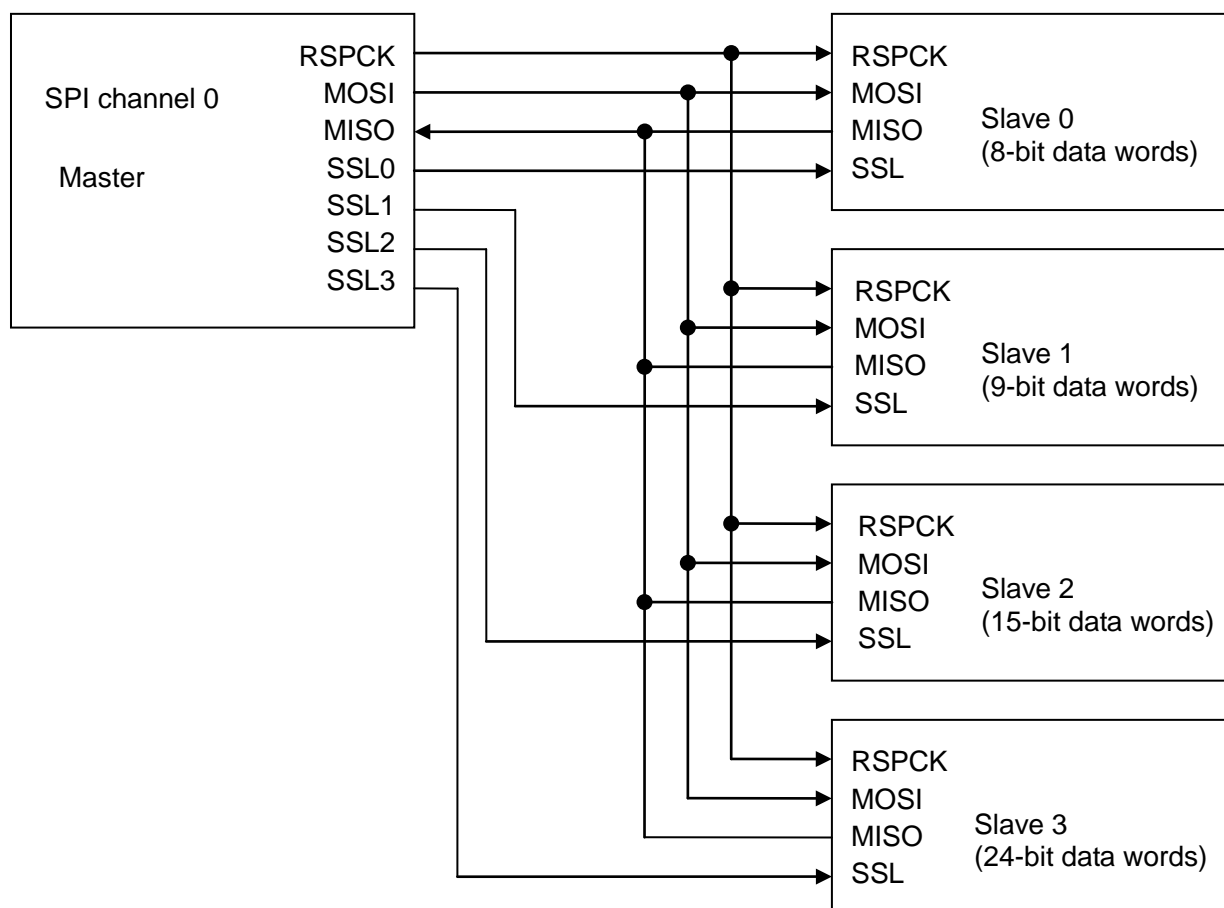


Figure 5-25 shows how data of appropriate bit lengths is transferred to each SPI slave. Commands 0 to 3 are executed in sequence, with each command asserting the appropriate SSL pin.

```
/* Peripheral driver function prototypes */
#include "r_pdl_cgc.h"
#include "r_pdl_spi.h"

/* PDL device-specific definitions */
#include "r_pdl_definitions.h"

#define MASTER_CHANNEL 0

void main(void)
{
    const uint32_t master_tx_data[4] = \
    {
        0x000000A4, /* 8-bit data */
        0x00000132, /* 9-bit data */
        0x00007F34, /* 15-bit data */
        0x00345678 /* 24-bit data */
    };

    uint32_t master_rx_data[4] = \
```

```

{
    0x00000000,
    0x00000000,
    0x00000000,
    0x00000000
};

/* Initialise the system clocks */
R_CGC_Set(
    12.5E6,
    100E6,
    50E6,
    PDL_NO_DATA
);

/* Configure the master SPI channel using -A pins */
R_SPI_Create(
    MASTER_CHANNEL,
    PDL_SPI_MODE_SPI_MASTER | PDL_SPI_PIN_A | \
    PDL_SPI_PIN_SSL0_LOW | PDL_SPI_PIN_SSL1_LOW | \
    PDL_SPI_PIN_SSL2_LOW | PDL_SPI_PIN_SSL3_LOW,
    PDL_SPI_FRAME_4,
    PDL_NO_DATA,
    2E6
);

/* Prepare the transfer with slave 0 */
R_SPI_Command(
    MASTER_CHANNEL,
    0,
    PDL_SPI_CLOCK_MODE_0 | PDL_SPI_LSB_FIRST | \
    PDL_SPI_ASSERT_SSL0 | PDL_SPI_LENGTH_8,
    PDL_NO_DATA
);

/* Prepare the transfer with slave 1 */
R_SPI_Command(
    MASTER_CHANNEL,
    1,
    PDL_SPI_CLOCK_MODE_0 | PDL_SPI_LSB_FIRST | \
    PDL_SPI_ASSERT_SSL1 | PDL_SPI_LENGTH_9,
    PDL_NO_DATA
);

/* Prepare the transfer with slave 2 */
R_SPI_Command(
    MASTER_CHANNEL,
    2,
    PDL_SPI_CLOCK_MODE_0 | PDL_SPI_LSB_FIRST | \
    PDL_SPI_ASSERT_SSL2 | PDL_SPI_LENGTH_15,
    PDL_NO_DATA
);

/* Prepare the transfer with slave 3 */
R_SPI_Command(
    MASTER_CHANNEL,
    3,
    PDL_SPI_CLOCK_MODE_0 | PDL_SPI_LSB_FIRST | \
    PDL_SPI_ASSERT_SSL3 | PDL_SPI_LENGTH_24,
    PDL_NO_DATA
);

/* Transfer all the data once */
R_SPI_Transfer(
    MASTER_CHANNEL,
    PDL_NO_DATA,
    master_tx_data,
    master_rx_data,

```

```
1,  
PDL_NO_FUNC,  
0  
);  
}
```

**Figure 5-25: Example of multiple slave Serial Peripheral Interface use**



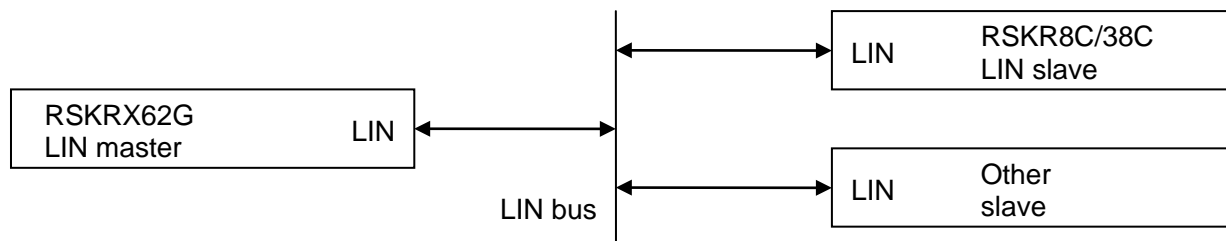
### 5.13. LIN module

The following examples show the use of the RX62G LIN module as a LIN master.

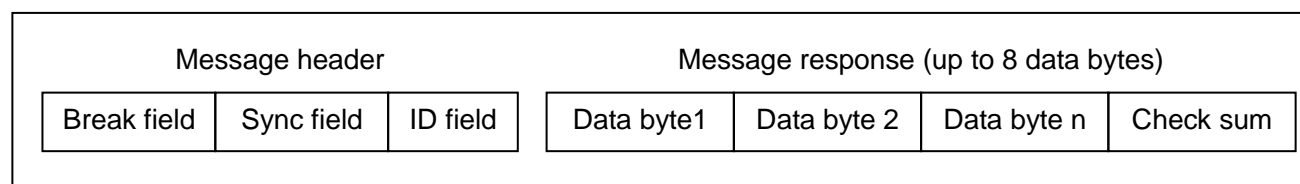
The RSK boards carry the MCU and also a LIN transceiver IC.

The transceiver combines the LTX and LRX signals into the single-wire LIN signal.

A typical connection is shown below.



A LIN transfer consists of a message header (always sent by the master), followed by a message response (sent by the master or a slave).



**Figure 5-26: LIN data transfer**

## 5.13.1. Transmit LIN data

Figure 5-27 shows how a message header with ID 31h and eight data bytes are transmitted by the LIN master. Polling is used to detect when the transfer has completed.

```

/* Peripheral driver function prototypes */
#include "r_pdl_cgc.h"
#include "r_pdl_lin.h"

/* PDL device-specific definitions */
#include "r_pdl_definitions.h"

void main(void)
{
    const uint8_t tx_data_store[8] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
    uint16_t StatusValue = 0;

    /* Initialise the system clocks */
    R_CGC_Set(
        12.5E6,
        100E6,
        50E6,
        PDL_NO_DATA
    );

    /* Configure LIN channel 0 */
    R_LIN_Create(
        0,
        PDL_LIN_ERROR_ALL_ENABLE,
        PDL_LIN_TB_LOW_13 | PDL_LIN_TB_HIGH_1,
        PDL_LIN_IBHR_SPACE_0 | PDL_LIN_IB_SPACE_0 | \
        PDL_LIN_WAKE_SPEC_2 | PDL_LIN_WAKE_LENGTH_1,
        19200,
        PDL_NO_FUNC,
        0
    );

    /* Send 8 bytes to ID 31h */
    R_LIN_Transfer(
        0,
        PDL_LIN_TX_DATA | \
        PDL_LIN_PARITY_CALCULATE | PDL_LIN_CHECKSUM_CLASSIC,
        0x31,
        tx_data_store,
        8
    );

    do
    {
        /* Read the status of channel 0 */
        R_LIN_GetStatus(
            0,
            &StatusValue,
            PDL_NO_PTR
        );
    } while ((StatusValue & BIT_3) != BIT_3);
}

```

Figure 5-27: Example of LIN master data transmission

## 5.13.2. Receive data from a LIN slave

The LIN slave is a Renesas R8C/38C MCU, programmed to transmit two bytes of data in response to ID code 11h.

A LIN transceiver IC fitted on the RSKRX62G board has a wake-up pin. This is connected to pin PA2 on the RX62G.

Figure 5-27 shows how a message header with ID 11h causes the R8C/38C to transmit 2 bytes.

The received data contains an ADC value from the R8C/38C.

Interrupts are used to detect the data reception.

```

/* Peripheral driver function prototypes */
#include "r_pdl_cgc.h"
#include "r_pdl_io_port.h"
#include "r_pdl_cmt.h"
#include "r_pdl_lin.h"

/* PDL device-specific definitions */
#include "r_pdl_definitions.h"

volatile bool transfer_in_progress;

/* LIN callback function */
void LIN_Event_handler(void);

void main(void)
{
    uint8_t rx_data_store[2];
    uint16_t adc_reading;

    /* Configure the clocks */
    R_CGC_Set(
        12.5E6,
        100E6,
        50E6,
        PDL_NO_DATA
    );

    /* Wake up the LIN transceiver */
    R_IO_PORT_Write(
        PDL_IO_PORT_A_2,
        1
    );
    R_IO_PORT_Set(
        PDL_IO_PORT_A_2,
        PDL_IO_PORT_OUTPUT
    );

    /* Configure LIN channel 0 */
    R_LIN_Create(
        0,
        PDL_LIN_ERROR_ALL_ENABLE,
        PDL_LIN_TB_LOW_13 | PDL_LIN_TB_HIGH_1,
        PDL_LIN_IBHR_SPACE_0 | PDL_LIN_IB_SPACE_0 | \
        PDL_LIN_WAKE_SPEC_2 | PDL_LIN_WAKE_LENGTH_1,
        19200,
        LIN_Event_handler,
        10
    );

    /* Allow 100µs for the LIN bus to settle */
    R_CMT_CreateOneShot(
        0,
        PDL_CMT_PERIOD,
        100E-6,
        PDL_NO_FUNC,

```

```

    0
);

transfer_in_progress = true;
/* Expect 2 bytes from the R8C/38C */
R_LIN_Transfer(
    0,
    PDL_LIN_RX_DATA | \
    PDL_LIN_PARITY_CALCULATE | PDL_LIN_CHECKSUM_CLASSIC,
    0x11,
    PDL_NO_PTR,
    2
);

while (transfer_in_progress == true);

/* Read the 2 bytes */
R_LIN_Read(
    0,
    rx_data_store,
    2
);

/* Calculate the ADC value */
adc_reading = (uint16_t)((rx_data_store[0] * 256) + rx_data_store[1]);
}

void LIN_Event_handler(void)
{
    uint16_t StatusValue = 0;

    /* Read the status of channel 0 */
    R_LIN_GetStatus(
        0,
        &StatusValue,
        PDL_NO_PTR
    );

    /* Receive complete? */
    if ((StatusValue & BIT_4) != 0x0u)
    {
        transfer_in_progress = false;
    }
}

```

Figure 5-28: Example of LIN master data reception

## 5.13.3. Self-test mode

Figure 5-29 shows how the transmission and reception of data is tested using self-test mode.

```

/* Peripheral driver function prototypes */
#include "r_pdl_cgc.h"
#include "r_pdl_lin.h"

/* PDL device-specific definitions */
#include "r_pdl_definitions.h"

static uint8_t calc_LIDB(const uint8_t);

volatile bool transfer_in_progress;

/* LIN callback function */
void LIN_Event_handler(void);

void main(void)
{
    const uint8_t tx_data_store[8] = {"LIN data"};
    uint8_t rx_data_store[8];
    uint8_t Received_checksum = 0;
    uint8_t reference_checksum_inverted_data;
    uint16_t temp16;
    uint8_t loop_counter;

    /* Configure the clocks */
    R_CGC_Set(
        12.5E6,
        100E6,
        50E6,
        PDL_NO_DATA
    );

    /* Calculate the checksum of the inverted data */
    temp16 = 0;
    for (loop_counter = 0; loop_counter <= 7; loop_counter++)
    {
        temp16 += (tx_data_store[loop_counter] ^ 0xFF);
        if (temp16 >= 256)
        {
            temp16 -= 256;
        }
    }
    reference_checksum_inverted_data = (uint8_t)temp16;

    /* Configure LIN channel 0 */
    R_LIN_Create(
        0,
        PDL_LIN_ERROR_ALL_ENABLE,
        PDL_LIN_TB_LOW_13 | PDL_LIN_TB_HIGH_1,
        PDL_LIN_IBHR_SPACE_0 | PDL_LIN_IB_SPACE_0 | \
        PDL_LIN_WAKE_SPEC_2 | PDL_LIN_WAKE_LENGTH_1,
        19200,
        LIN_Event_handler,
        10
    );

    /* Select self-test mode */
    R_LIN_Control(
        0,
        PDL_LIN_MODE_SELFTEST,
        PDL_NO_DATA
    );
}

```

```

transfer_in_progress = true;
/* Send 8 bytes to ID 31h */
R_LIN_Transfer(
    0,
    PDL_LIN_TX_DATA | \
    PDL_LIN_PARITY_CALCULATE | PDL_LIN_CHECKSUM_CLASSIC,
    0x31,
    tx_data_store,
    8
);
while (transfer_in_progress == true);

/* Read the status of channel 0 */
R_LIN_GetStatus(
    0,
    PDL_NO_DATA,
    &Received_checksum
);
/* In self-test mode, the checksum is inverted when read */
Received_checksum ^= 0xFF;

/* Check the transmitted ID and parity */
if ( (LIN0.LIDB.BYTE ^ 0xFF) != calc_LIDB(0x31) )
{
    while(1);
}

/* Check the check sum */
if (Received_checksum != reference_checksum_inverted_data)
{
    while(1);
}

/* Write to the check sum register */
R_LIN_Control(
    0,
    PDL_LIN_CHECKSUM_WRITE,
    reference_checksum_inverted_data
);
/* Reload the transmit data */
LIN0.LDB1 = tx_data_store[0];
LIN0.LDB2 = tx_data_store[1];
LIN0.LDB3 = tx_data_store[2];
LIN0.LDB4 = tx_data_store[3];
LIN0.LDB5 = tx_data_store[4];
LIN0.LDB6 = tx_data_store[5];
LIN0.LDB7 = tx_data_store[6];
LIN0.LDB8 = tx_data_store[7];

transfer_in_progress = true;
/* Read 8 bytes from ID 15 */
R_LIN_Transfer(
    0,
    PDL_LIN_RX_DATA | \
    PDL_LIN_PARITY_CALCULATE | PDL_LIN_CHECKSUM_CLASSIC,
    15,
    PDL_NO_PTR,
    8
);
while (transfer_in_progress == true);

/* Read 8 bytes */
R_LIN_Read(
    0,
    rx_data_store,
    8
);

```

```

/* Check that received data is the inverted version of the original */
for (loop_counter = 0; loop_counter <= 7; loop_counter++)
{
    if ( (rx_data_store[loop_counter] ^ 0xFF) != tx_data_store[loop_counter])
    {
        while(1);
    }
}

void LIN_Event_handler(void)
{
    uint16_t StatusValue = 0;

    /* Read the status of channel 0 */
    R_LIN_GetStatus(
        0,
        &StatusValue,
        PDL_NO_PTR
    );

    if ((StatusValue & (BIT_4 | BIT_3)) != 0x0u)
    {
        transfer_in_progress = false;
    }
}

static uint8_t calc_LIDB(const uint8_t id_value)
{
    uint8_t p0;
    uint8_t p1;
    uint8_t lldb_copy;

    /* Calculate parity bit P0 */
    p0 = id_value;
    p0 ^= (uint8_t)(id_value >> 1);
    p0 ^= (uint8_t)(id_value >> 2);
    p0 ^= (uint8_t)(id_value >> 4);
    p0 &= 0x01u;

    /* Calculate parity bit P1 */
    p1 = (uint8_t)(id_value >> 1);
    p1 ^= (uint8_t)(id_value >> 3);
    p1 ^= (uint8_t)(id_value >> 4);
    p1 ^= (uint8_t)(id_value >> 5);
    p1 &= 0x01u;

    /* Build the register L0IDB */
    lldb_copy = (uint8_t)((p1 << 7) | (p0 << 6) | (id_value & 0x03F));

    return lldb_copy;
}

```

Figure 5-29: LIN master self-test example

### 5.14. 12-bit Analog to Digital Converter

Figure 5-30 shows an example of 12-bit ADC usage. Interrupts are enabled for ADC, which operates in single mode. The CPU sleeps during the conversion.

```

/* Peripheral driver function prototypes */
#include "r_pdl_adc_12.h"
#include "r_pdl_cgc.h"
#include "r_pdl_intc.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

volatile uint8_t cmp_flag;
volatile uint16_t adc0_result;
void ADC_12_0_Callback(void);

void main(void)
{
    /* Initialise the system clocks */
    R_CGC_Set(
        12.5E6,
        100E6,
        50E6,
        PDL_NO_DATA
    );

    /* Set the CPU's Interrupt Priority Level to 0 */
    R_INTC_Write(
        PDL_INTC_REG_IPL,
        0
    );

    /* Configure the ADC unit for single scan on pin AN000 with interrupt enabled */
    R_ADC_12_CreateUnit(
        0,
        PDL_ADC_12_MODE_SINGLE | PDL_ADC_12_CHANNELS_OPTION_0 | \
        PDL_ADC_12_ADSSTR_CALCULATE,
        PDL_NO_DATA,
        PDL_NO_DATA,
        10E6,
        20E-6,
        ADC_12_0_Callback,
        6
    );

    /* Set gain for AN000 and comparator for AN100 */
    R_ADC_12_CreateChannel(
        PDL_ADC_12_CMP_REFH_AVCC0_4_8 | PDL_ADC_12_CMP_AN10X_REFH_INT,
        PDL_ADC_12_CH_GAIN_2_000,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_ADC_12_CH_CMP_HIGH,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_FUNC,
        0
    );

    /* Start conversions on ADC unit 0 */
    R_ADC_12_Control(
        PDL_ADC_12_0_ON | PDL_ADC_12_CPU_OFF
    );

    /* Shutdown ADC unit */
    R_ADC_12_Destroy(
        PDL_ADC_12_0_DESTROY | PDL_ADC_12_CMP_100_DESTROY
    );
}

```



```
    );  
}  
  
void ADC_12_0_Callback(void)  
{  
    /* Read the level on AN1 */  
    R_ADC_12_Read(  
        0,  
        &cmp_flag,  
        &adc0_result,  
        &adc0_result  
    );  
}
```

**Figure 5-30: Example of 12-bit ADC Conversion**

### 5.15. 10-bit Analog to Digital Converter

Figure 5-31 shows an example of 10-bit ADC usage. Interrupts are enabled for the ADC, which operates in single mode. CPU sleeps during the conversion.

```

/* Peripheral driver function prototypes */
#include "r_pdl_adc_10.h"
#include "r_pdl_cgc.h"
#include "r_pdl_intc.h"

/* RPDL device-specific definitions */
#include "r_pdl_definitions.h"

volatile uint16_t adc0_result;
void ADC0Handler(void);

void main(void)
{
    /* Initialise the system clocks */
    R_CGC_Set(
        12.5E6,
        100E6,
        50E6,
        PDL_NO_DATA
    );

    /* Set the CPU's Interrupt Priority Level to 0 */
    R_INTC_Write(
        PDL_INTC_REG_IPL,
        0
    );

    /* Configure the ADC unit for single scan on pin AN0 with interrupt enabled */
    R_ADC_10_Create(
        0,
        PDL_ADC_10_MODE_SINGLE | PDL_ADC_10_CHANNELS_OPTION_0,
        PDL_ADC_10_TRIGGER_SOFTWARE,
        12E6,
        20E-6,
        ADC0Handler,
        6
    );

    /* Start conversions on ADC unit*/
    R_ADC_10_Control(
        PDL_ADC_10_0_ON | PDL_ADC_10_CPU_OFF
    );

    /* Shutdown ADC unit */
    R_ADC_10_Destroy(
        0
    );
}

void ADC0Handler(void)
{
    /* Read the level on AN1 */
    R_ADC_10_Read(
        0,
        &adc0_result
    );
}

```

Figure 5-31: Example of 10-bit ADC Conversion

## 6. RX-specific notes

### 6.1. Interrupts and processor mode

The RX CPU has two processor modes; supervisor and user.

The API driver functions will be executed by the CPU in either mode.

However, any callback functions which are called by the API interrupt handlers 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 callback function and any function that is called by the callback 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.

### 6.2. Interrupts and DSP instructions

The accumulator (ACC) register is modified by the following instructions:

- i. DSP (MACHI, MACLO, MULHI, MULLO, MVTACHI, MVTACLO and RACW).
- ii. Multiply and multiply-and-accumulate (EMUL, EMULU, FMUL, MUL, and RMPA)

The accumulator (ACC) register is not pushed onto the stack by the API interrupt handlers.

If DSP instructions are being utilised in the users' code, callback functions which are called by the API interrupt handlers should either

- a) Avoid using instructions which modify the ACC register.
- b) Take a copy of the ACC register and restore it before exiting the callback function.

Revision History	RX62G, RX62T Group User's Manual
------------------	----------------------------------

Rev.	Date	Description	
		Page	Summary
1.00	Sep. 11, 2012	—	First issue. Developed using RX62G hardware manual Rev.1.00 and RX62T hardware manual Rev.1.30.

---

Renesas Peripheral Driver Library  
User's Manual  
RX62G, RX62T Group

Publication Date: Rev.1.00 Sep 11, 2012

Published by: Renesas Electronics 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: +44-1628-651-700, Fax: +44-1628-651-804

### **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.**

13F, No. 363, Fu Shing North Road, Taipei, Taiwan  
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

### **Renesas Electronics Singapore Pte. Ltd.**

80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre Singapore 339949  
Tel: +65-6213-0200, Fax: +65-6213-0300

### **Renesas Electronics Malaysia Sdn.Bhd.**

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

### **Renesas Electronics Korea Co., Ltd.**

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

RX62G, RX62T Group