User's Manual

32

Renesas Peripheral Driver Library

User's Manual

RX220 Group

Renesas Electronics
www.renesas.com

Rev.1.11 Aug 2014

Notice

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

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

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

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

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

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

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 document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins

   Handle unused pins in accordance 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 a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Table of Contents

1. Introduction

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

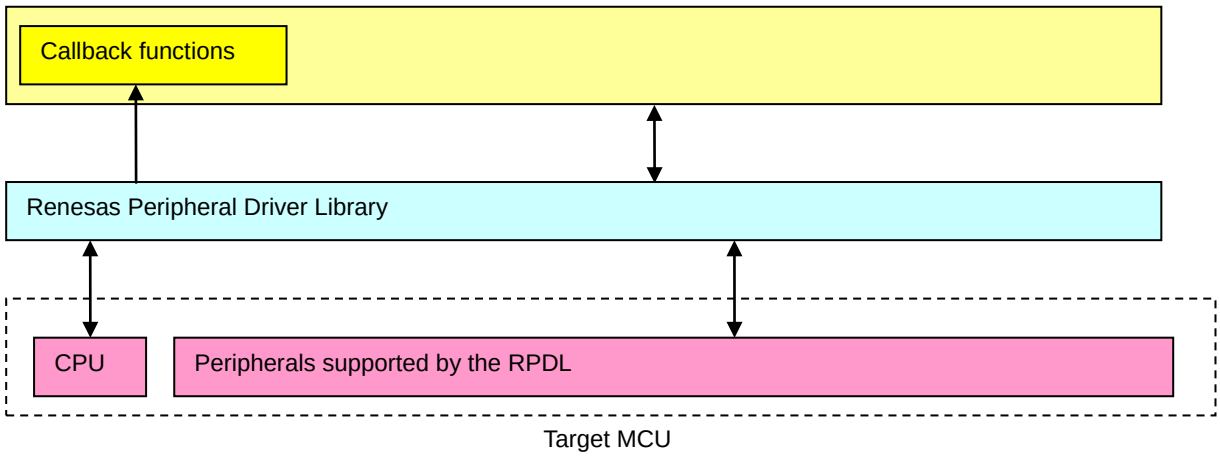


Figure 1: System configuration, with all peripherals supported by RPDL

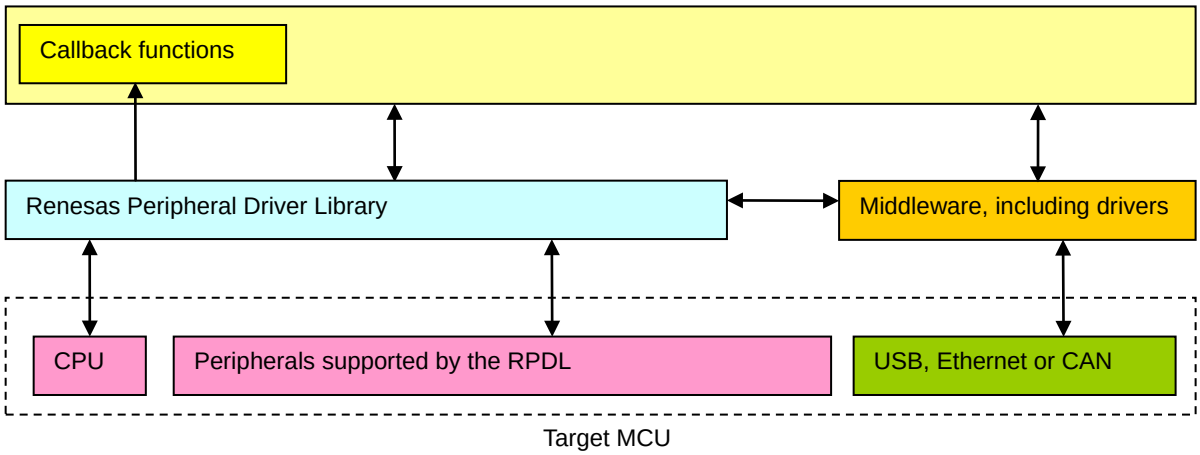


Figure 2: System configuration, with middleware taking direct control of some peripherals

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

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

RPDL has not been designed to be compatible for use with an RTOS.

The coding standards and naming conventions are specified by Renesas.

1.1. Tool chain requirements

This RPDL 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. Compiler options when you use this product

(1) The options which must be specified in your project are listed below.
The options other than -cpu, -dbl_size are the default setting of the compiler.

-cpu = rx200
-round = nearest
-denormalize = off
-dbl_size = 8
-unsigned_char
-unsigned_bitfield
-bit_order = right
-unpack
-noexception
-rtti = off
-fint_register = 0
-branch = 24

(2) The options which must NOT be specified in your project are listed below.
As the default setting of the compiler, the following options are not specified.

-int_to_short
-auto_enum
-base
-patch
-pic
-pid
-nouse_pid_register
-save_acc

1.3. Using the library within your project

The driver library can be used in two ways.

1.3.1. Via the PDG graphical utility

PDG can be downloaded from www.renesas.com/pdg.
The directions for use of the PDG utility are given in the PDG manual.

1.3.2. Using RPDL stand-alone

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

a) Unzip the RPDL 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.

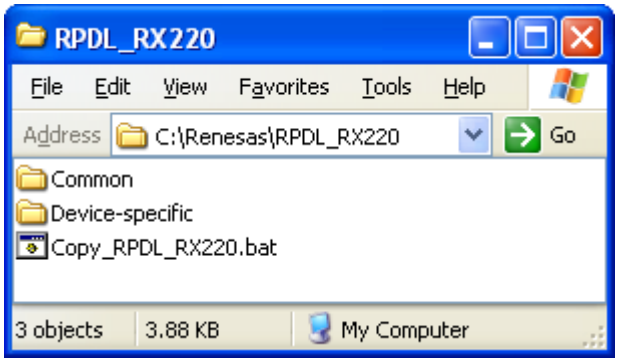
The instructions to follow for stand-alone use start are given below.

1) Unzip the RPDL files

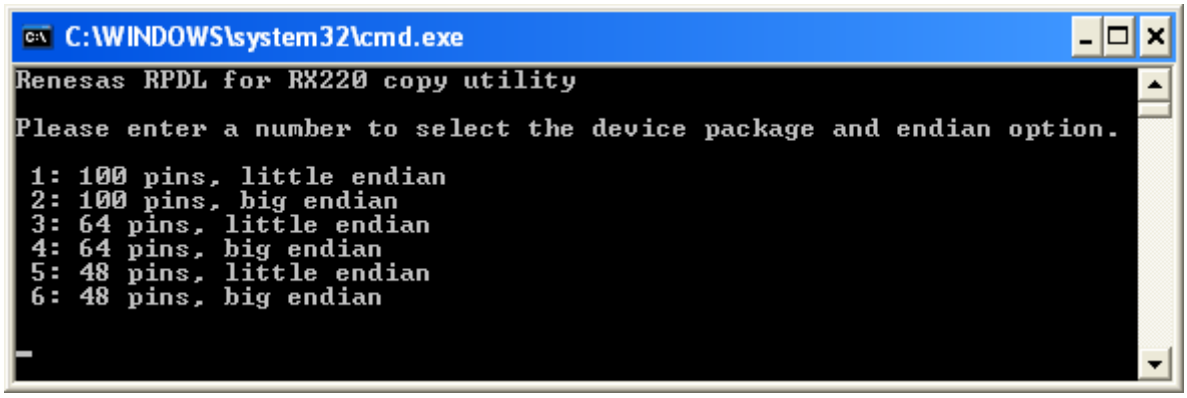
Double-click on the file RPDL_RX220.exe to unpack the files.
The default location is C:\Renesas\RPDL_RX220.

2) Copy the files into your project area

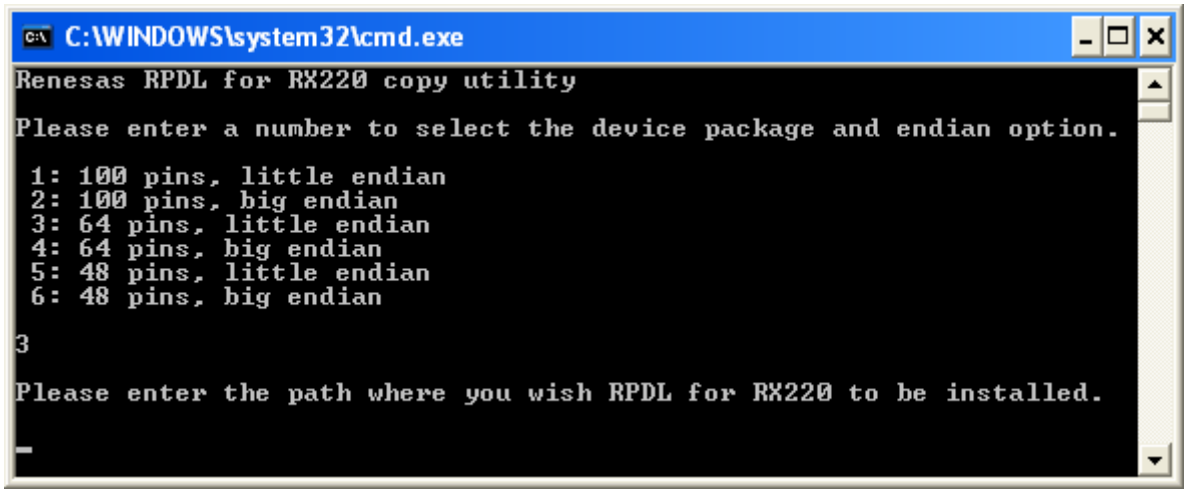
Navigate to where the RPDL files were unpacked.



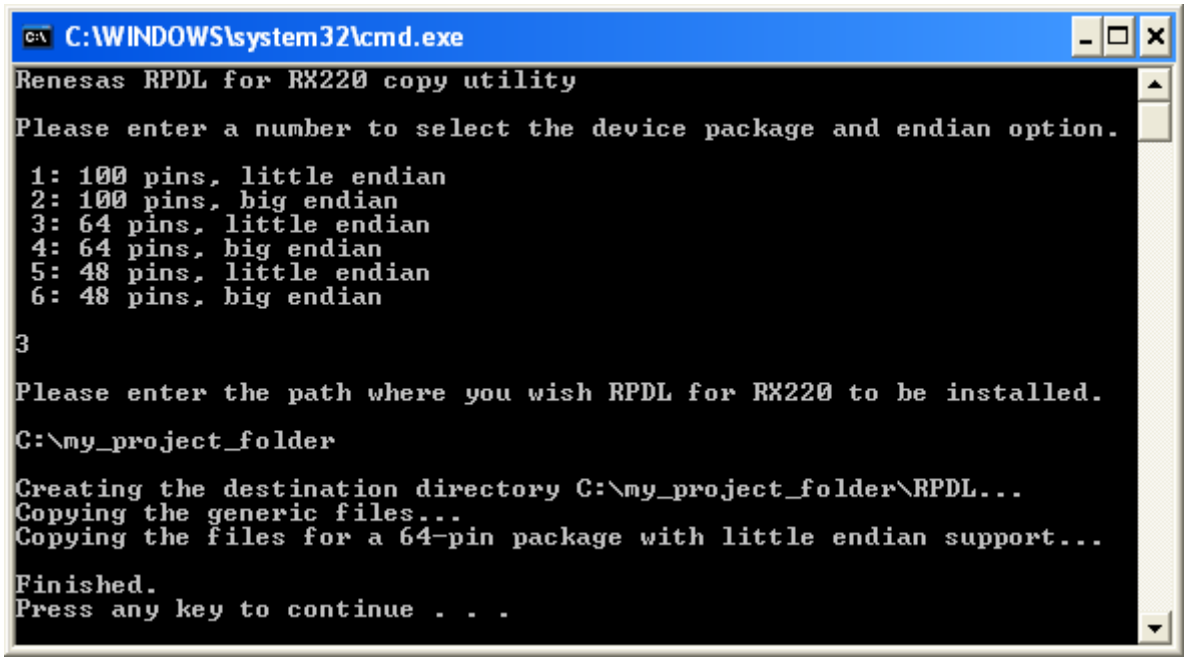
Double-click on “Copy_RPDL_RX220.bat” to start the copy process.



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



Type the full path to the folder where you wish RPDL 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.



Press any key to close the window.
Copy folder “\RPDL” into the folder project workspace created. (Example “C:\WorkSpace\rpdl_lib_test\ rpdl_lib_test”).

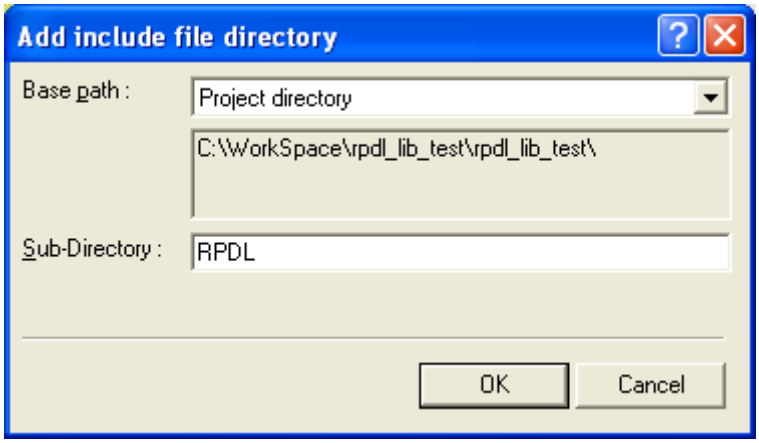
3) Include the new directory

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:

x

Click on “OK” to close the window.

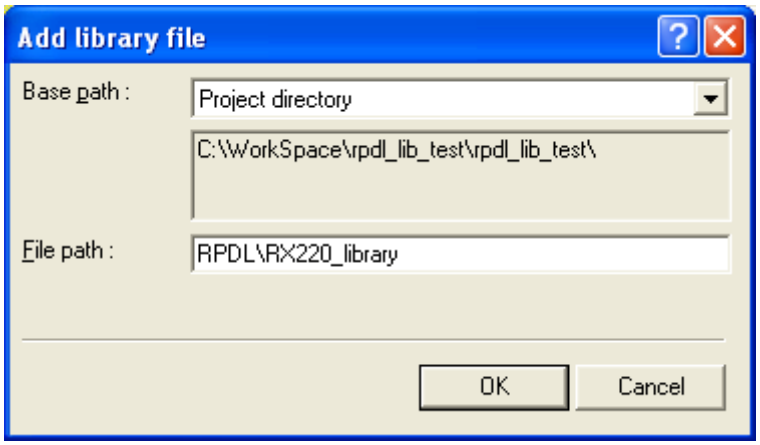
4) Add the RPDL library file

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

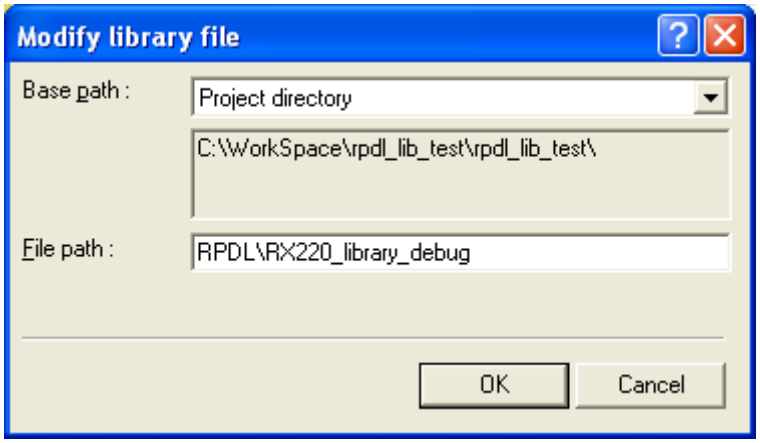
Select the Link/Library tab.

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\RX220_library” as the File path.



To use library with debug information, enter “RPDL\RX220_library_debug” as the File path.



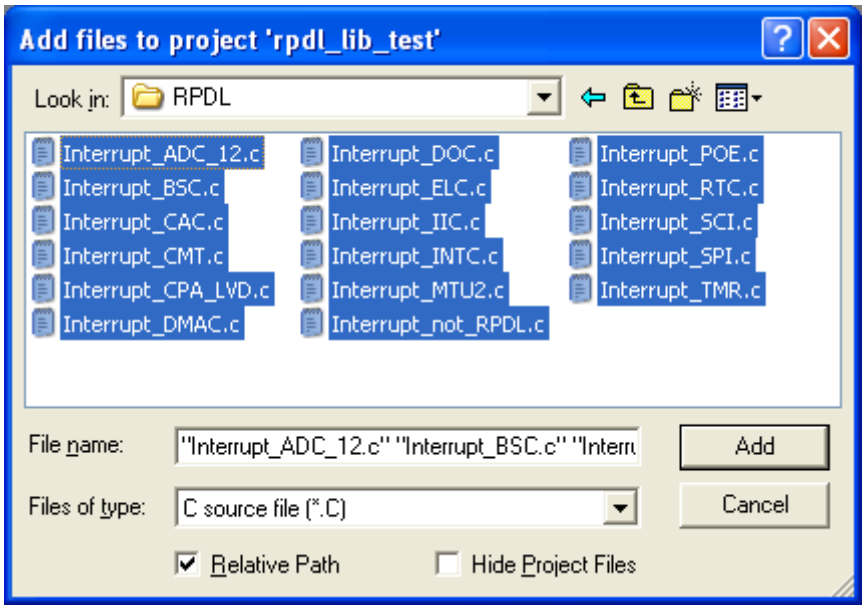
Click on “OK” to close the window.
Click on “OK” to return to the main HEW window.

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 RPDL 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).

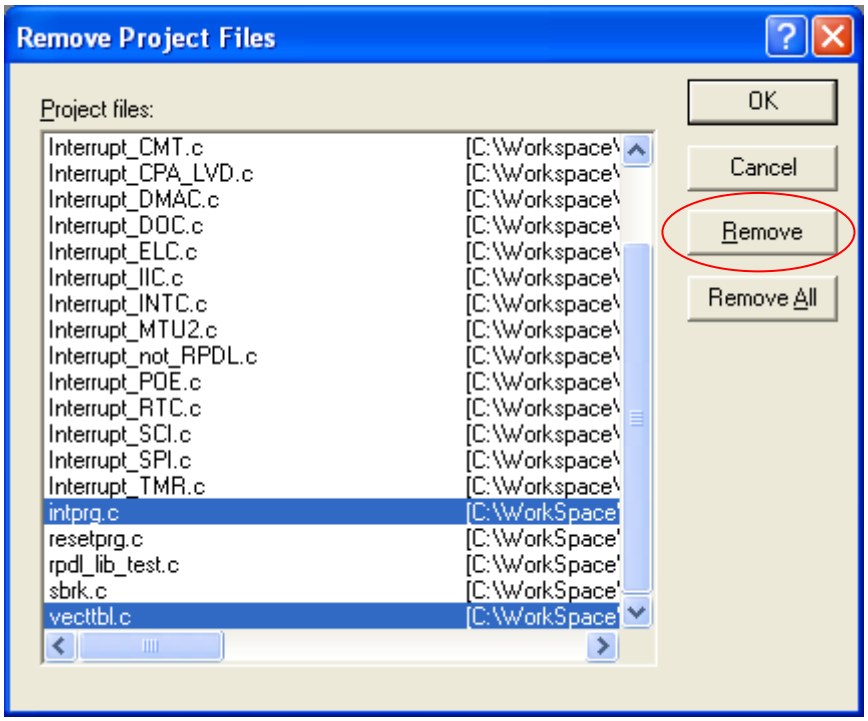
8) Avoid conflicts with standard project files

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

(a) Removal

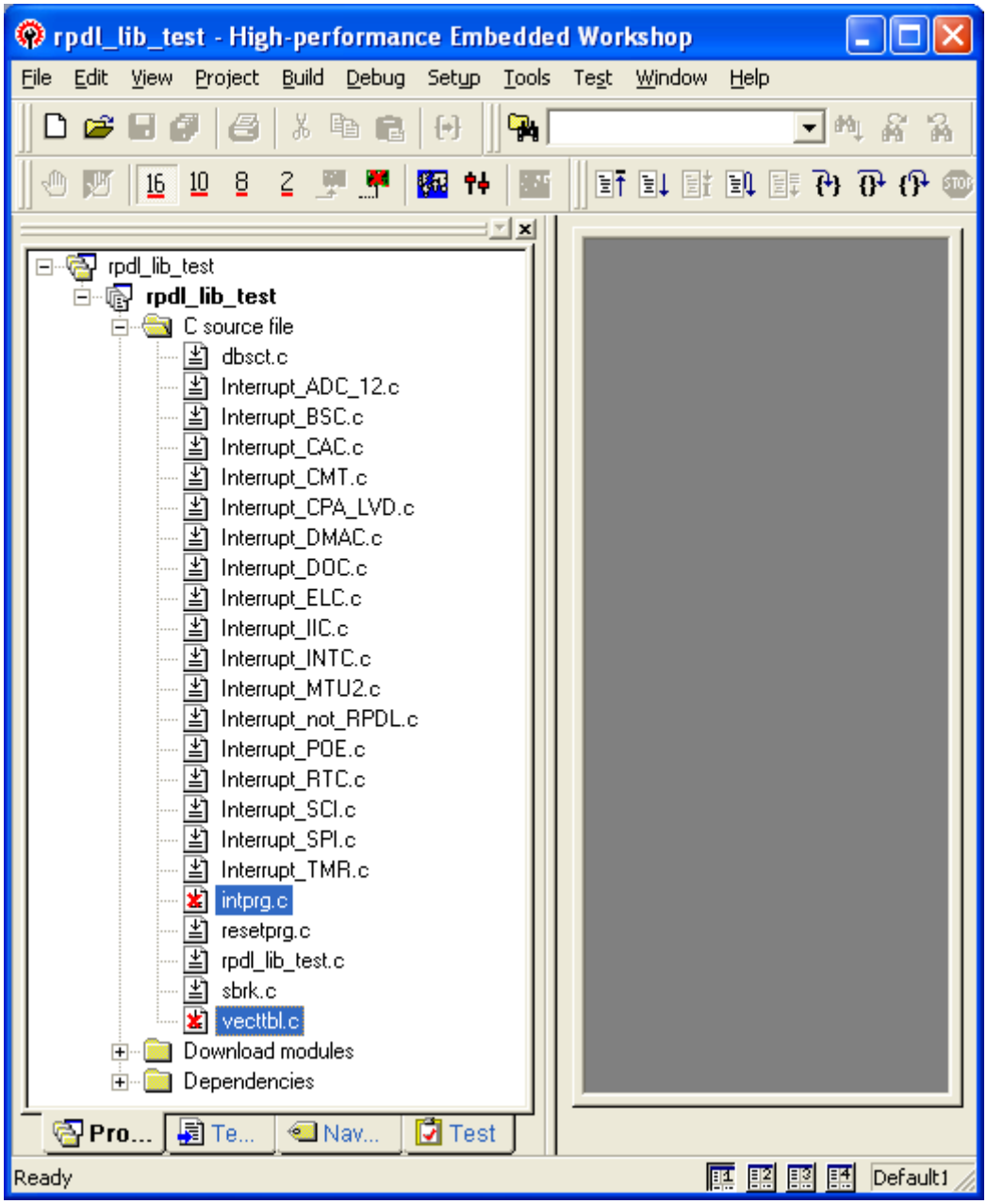
Use the key sequence Alt, P, R to open the “Remove Project Files” window.

Select the files and click on Remove.



(b) Exclusion

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



9) Set the build options.

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

In this section, only options which you must change from the default settings are described. If you add RPDL in existing project, see also “1.2 Compiler options when you use this product”.

(a) Set the optimisation

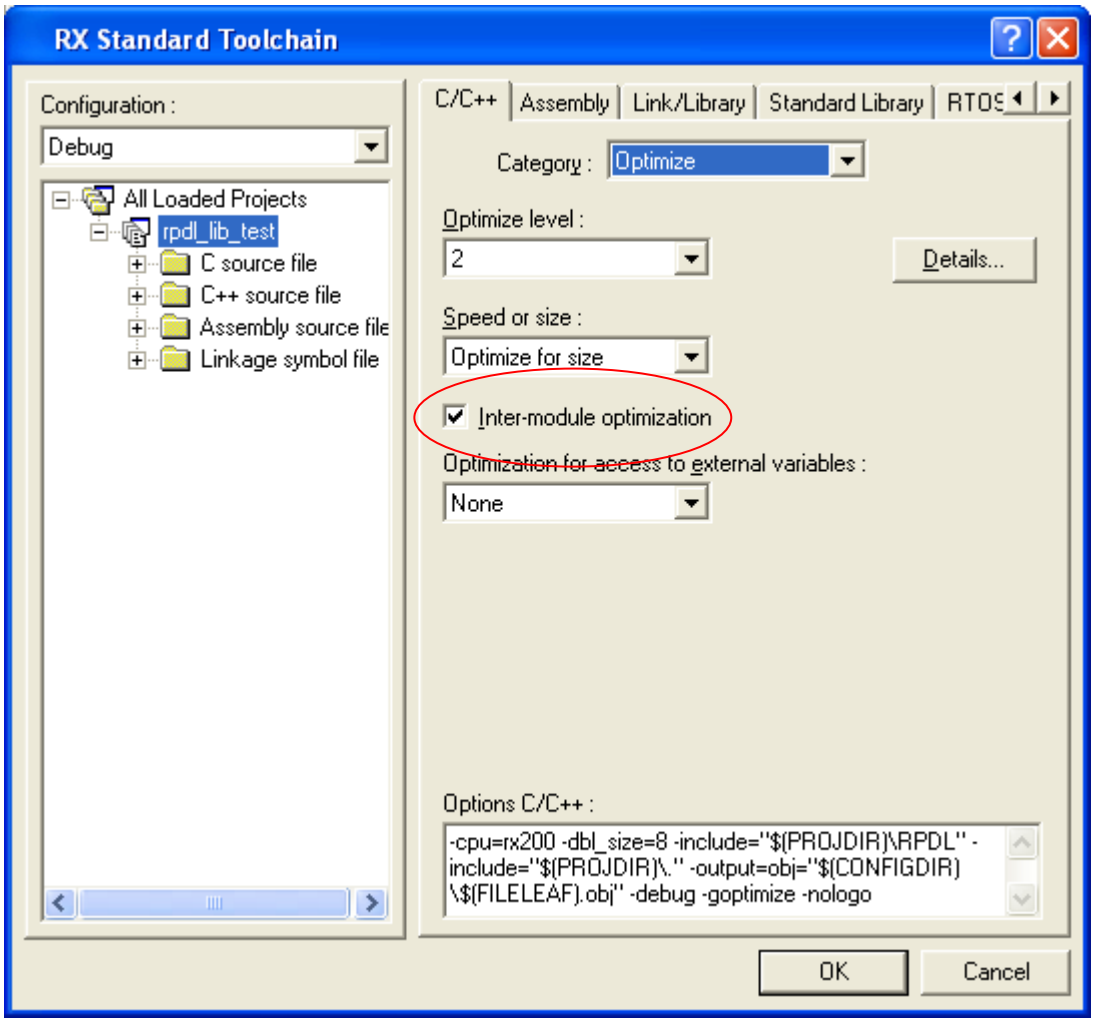
To avoid linking unused RPDL functions, adjust the Compiler and Linker settings.

(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.

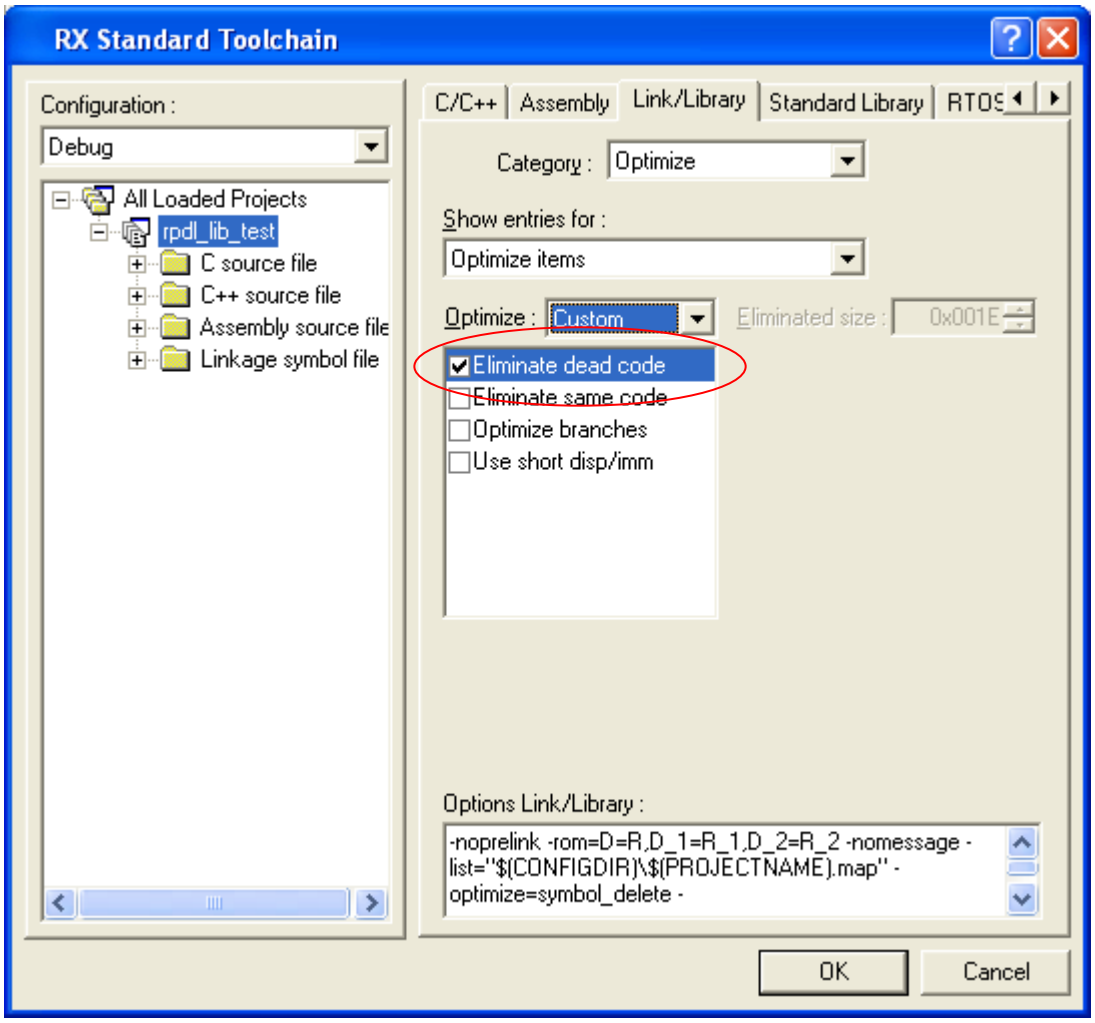


(ii) Linker

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.



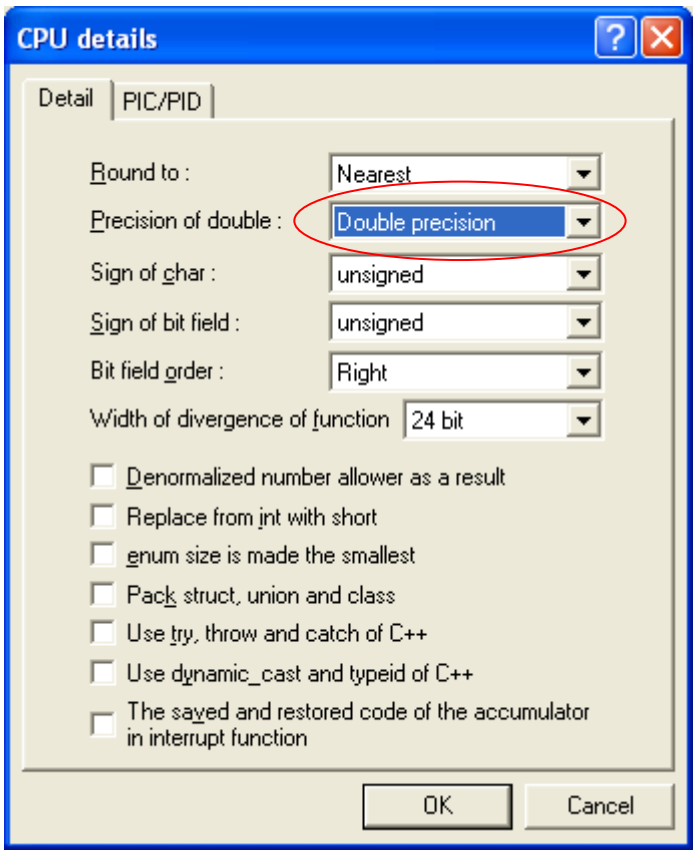
(b) Set the floating point precision

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.

11) Using library with debug information

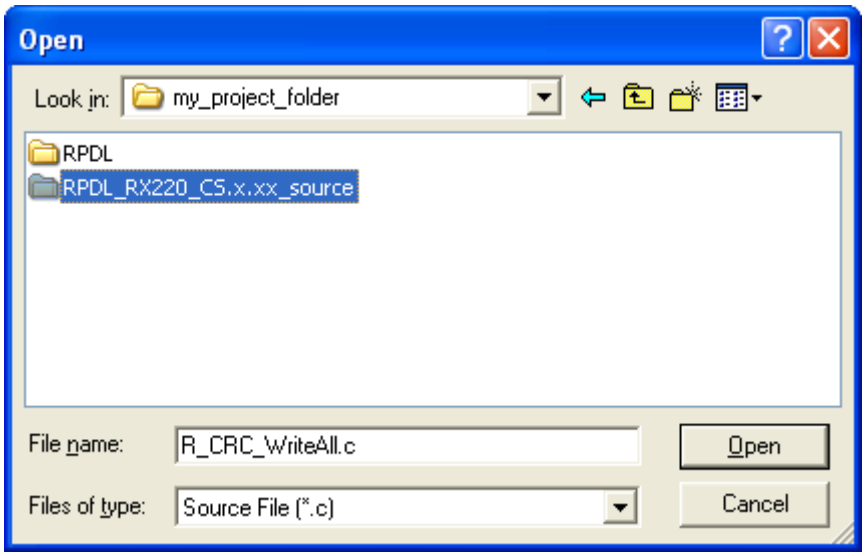
RPDL library with debug information should be chosen, in order to step in the RPDL source code for debugging.

Unzip the RPDL source zip file (e.g. “RPDL_RX220_CS.x.xx_source.zip”) into a folder (e.g. “C:\my_project_folder”).

Set a breakpoint at the RPDL API to be debugged.

When the program breaks at the RPDL API, press “F11” key to step in the function.

A pop-up window will appear to request for the location of the corresponding RPDL source file.



Select the folder where you unzip the RPDL source file, and open the source file under respective module folder.

Once the correct source file is selected, user could step in to the file and step through the function.

1.3.3. Header file inclusion

The RPDL folder contains a header file, iodefine_RPDL.h.
This file is included by the RPDL source files and will also be included by any user-generated files that call RPDL functions.

The main HEW project folder may contain the header file iodefine.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 RPDL functions, there is no need to include this file iodefine.h.

1.3.4. 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"
```

1.3.5. Recommended initialisation code

The RX tool chain has a designated function for MCU initialisation, HardwareSetup().

During the MCU initialisation phase, it is recommended that the following functions are placed in this function.

Note that the file resetprg.c (supplied when a new project is created) requires editing to remove the "//" comment identifiers for the two lines below.

```
//extern void HardwareSetup(void);
//      HardwareSetup();
```

1) Initialisation of pins that are not available

For pins that are not available on the selected MCU package type, set the control registers to the recommended values using

```
R_IO_PORT_NotAvailable();
```

This function can be called even if the largest device has been selected. This will allow for the user's code to be ported to another project that does use a smaller MCU package.

2) Initialisation of the sub-clock oscillator if not used

If the sub-clock oscillator will not be used, it should be put into a stable state.
Please refer the program in Section 5.16.2

1.4. 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.

1.5. List of Abbreviations and Acronyms

Abbreviation	Full form
ADC	Analog to Digital Converter
API	Application Programming Interface
BCD	Binary-Coded Decimal
Bit	Binary digit
bps	Bits per second
BSC	Bus State Controller
CAN	Controller Area Network
CGC	Clock Generation Circuit
CMOS	Complementary Metal-Oxide Semiconductor
CMT	Compare Match Timer
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DAC	Digital to Analog Converter
DC	Direct Current
DMA	Direct Memory Access
DMAC	DMA Controller
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
HOCO	High-speed On-Chip Oscillator
I²C	Inter-Integrated Circuit
INTC	Interrupt Controller
I/O	Input / Output
IWDT	Independent WDT
kB	Kilo Byte (1024 bytes)
LOCO	Low-speed On-Chip Oscillator
LPC	Low Power Consumption
LSB	Least-Significant Bit
MB	Mega Byte (1024 kB)
MCU	Microcontroller Unit
MPC	Multifunction Pin Controller
MSB	Most-Significant Bit
MTU	Multi-function Timer pulse Unit
NMI	Non-Maskable Interrupt
OFS	Option Function Select
PDG	Peripheral Driver Generator
PLL	Phase-Locked Loop
POE	Port Output Enable
PPG	Programmable Pulse Generator
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
USB	Universal Serial Bus
WDT	Watchdog Timer

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) Multifunction Pin Controller
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) Voltage Detection Circuit
These driver functions are used for configuring the low-voltage detection response.

(7) Clock Frequency Accuracy Measurement Circuit
These driver functions are used for configuring and using clock measurement.

(8) Low Power Consumption
These driver functions are used for selecting lower power consumption.

(9) Register Write Protection
These driver functions are used for controlling access to protected registers.

(10) Bus Controller
These driver functions are used for configuring the external address bus, data bus and chip select pins and handling any bus errors.

(11) DMA Controller
These driver functions are used for configuring and controlling the transfer of data within the address space.

(12) Data Transfer Controller
These driver functions are used for configuring and controlling the transfer of data triggered by peripheral interrupts.

(13) Event Link Controller
These driver functions are used for configuring and controlling the event links.

(14) Multi-Function Timer Pulse Unit
These driver functions are used for configuring and controlling the multi-function timers.

(15) Port Output Enable
These driver functions are used for additional configuring and controlling of the timer outputs.

(16) 8-bit Timer
These driver functions are used for configuring and controlling the timers.

(17) Compare Match Timer
These driver functions are used for configuring and controlling the timers.

(18) Real-time Clock
These driver functions are used for configuring and controlling the real-time clock timer.

(19) Independent Watchdog Timer
These driver functions are used for configuring and controlling the timer.

(20) Serial Communication Interface
These driver functions are used to configure the serial channels and manage the transmission and / or reception of data across them.

(21) I²C Bus Interface
These driver functions are used for controlling the I²C bus channels.

(22) Serial Peripheral Interface
These driver functions are used for controlling the SPI channels.

(23) CRC calculator
These driver functions are used for controlling the calculator.

(24) 12-bit Analog to Digital Converter
These driver functions are used for configuring the 12-bit ADC units, controlling the units and reading the conversion results.

(25) Comparator A
These driver functions are used for configuring the Comparator A module.

(26) Data Operation Circuit
These driver functions are used for configuring the Data Operation Circuit module.

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 operation.
2. Controlling the clock generator operation.
3. 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. Selecting the applicable interrupt pins.
2. Configuration of an external interrupt signal for use.
3. Enabling use of the software interrupt.
4. Assigning an interrupt to be processed using the Fast Interrupt route.
5. Assigning handlers for the fixed exception interrupts.
6. Controlling an external interrupt input.
7. Reading the status of an external interrupt.
8. Reading an interrupt register.
9. Writing to an interrupt register.
10. 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.
9. Configuring the pins that are not available on smaller packages to the required state.

2.6. Multifunction Pin Controller Driver

The driver functions support access to the Multifunction Pin Controller (MPC) registers which select the mode of operation for some I/O pins.

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

1. Reading from an MPC register.
2. Writing to an MPC register.
3. Modifying an MPC 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 MCU features and on-chip ROM.
2. Reading the MCU status flags.
3. Setting the MCU start-up options.

2.8. Voltage Detection Circuit Driver

The driver function supports configuration of VDET1 and VDET2 voltage detection circuits. This function supports:

1. Configuration of the voltage detection circuit, including:
    - Setting voltage thresholds.
    - Defining a voltage event.
    - Configuring a reset when supply voltage drops below a voltage threshold.
2. Controlling the circuit operation.
3. Reading the circuit status.

2.9. Clock Frequency Accuracy Measurement Circuit Driver

The driver functions support access to the registers which control the Clock Frequency Accuracy Measurement Circuit. These functions support:

1. Configuring the operation.
2. Stopping the operation.
3. Modifying the operation.
4. Reading the status.

2.10. 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. Determining the cause of the exit from the lowest power mode.

2.11. Register Write Protection Driver

The driver functions support the control of the Register Write Protection, providing the following operations.

1. Enabling or disabling writing to the registers.
2. Reading the status of the write protection.

2.12. Bus Controller Driver

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

1. Setting the internal bus operation.
2. Configuration of the controller.
3. Controlling the bus controller.
4. Reading the status of the controller.

2.13. DMA Controller Driver

The driver functions support the control of the Direct Memory Access (DMA) controller, providing the following operations.

1. Configuration for use, including
   - Access to all control bits.
   - Automatic interrupt control
2. Disabling DMA channels that are no longer required and enabling low-power mode.
3. Control of a channel.
4. Reading the status and operation registers of a channel.

2.14. 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, stopping or modifying the operation of the controller.
5. Reading the status flags and data transfer registers.

2.15. Event Link Controller

The driver functions support the control of the Event Link Controller, providing the following operations.

1. Enabling the module.
2. Disabling the module.
3. Linking events with modules.
4. Configuring Timer output.
5. Setting and controlling port groups.

2.16. Multi-Function Timer Pulse Unit Driver

The driver functions support the use of the six 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.

2.17. 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.

2.18. 8-bit Timer Driver

The driver functions support the use of the four 8-bit timers, providing the following operations.

1. Selection of the TMR pins for use.
2. Configuring a channel for use, using register values which have been determined elsewhere.
3. Configuring two channels as a 16-bit pair, using register values which have been determined elsewhere.
4. Configuration for as a periodic timer, including
   - Automatic clock setting using frequency or period as an input.
   - Automatic pulse width setting, using pulse width or duty cycle as an input.
   - Automatic interrupt control
5. Configuration for as a one-shot timer, including
   - Automatic clock setting, using pulse width as an input
   - Automatic interrupt control
   - CPU sleep option
   - Automatic support for using two channels as a single 16-bit timer.
6. Disabling channels that are no longer required and enabling low-power mode.
7. Control of a single timer channel.
8. Control of two timer channels when configured as one 16-bit channel.
9. Control of channels in periodic mode, enabling pulse-width modulation (PWM) output.
10. Reading the registers of a single timer channel.
11. Reading the registers of a 16-bit timer channel pair.

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

2.19. 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.

2.20. Real-time Clock Driver

The driver functions support the use of the real-time clock, providing the following operations.

There are 2 count modes: calendar count mode and binary count mode.

For calendar count mode

1. Configuring the clock for use, including
   - Alarm configuration.
   - Optional day-of-week calculation.
   - 12 or 24 hour mode selection.
   - Automatic alarm and periodic interrupt control.
2. Disabling the clock.
3. Control of the clock, including
   - Changing the alarm settings.
   - Changing the current date or time.
   - Error adjustment.
4. Reading the clock status flags, current time and date, alarm time and date.
5. Reconfigure callback function and priority setting of alarm and periodic interrupts at warm start up.

For binary count mode

1. Configuring the count for use, including
   - Set current count.
   - Set alarm count.
   - Set alarm mask.
2. Disabling the count.
3. Control of the count, including
   - Changing the current count value.
   - Changing the alarm count value.
   - Changing the alarm mask value.
   - Error adjustment.
4. Reading the count status flags, current count, alarm count and alarm mask.
6. Reconfigure callback function and priority setting of alarm and periodic interrupts at warm start up.

2.21. 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.

2.22. Serial Communication Interface Driver

The driver functions support the use of the serial communication (SCI) channels providing the following operations.

1. Selection of the SCI pins for use.
2. Configuration for use, including
   - Automatic baud rate clock calculations
   - Automatic interrupt control
   - Automatic I/O pin configuration
   - Supporting the following modes:
     - Asynchronous
     - Multi-Processor
     - Clock Synchronous
     - Smart Card Interface
     - Simple IIC
     - Simple SPI
3. Disabling channels that are no longer required.
4. Transmitting data, with polling or interrupt mode automatically selected.
5. Receiving data, with polling or interrupt mode automatically selected.
6. Transmitting and/or receiving data in SPI mode, with polling or interrupt mode automatically selected.
7. Transmitting data in simple IIC mode, with polling or interrupt mode automatically selected.
8. Receiving data in simple IIC mode, with polling or interrupt mode automatically selected.
9. Transmitting the last byte of data in simple IIC mode.
10. Control the channel operation.
11. Reading the status flags.

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

2.23. I²C Bus Interface Driver

The driver functions support the use of the I²C module, providing the following operations.

1. Configuration for use, including
   - Automatic clock setting using transfer rate as an input.
   - Automatic interrupt control
2. Disabling the module that is 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 the unit, including bus lock-up recovery support.
9. Reading the status of the module.

Note: The Clock Generation Circuit must be configured before configuring the I²C module.

2.24. Serial Peripheral Interface Driver

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

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

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

2.25. 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.

2.26. 12-bit Analog to Digital Converter Driver

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

1. Selection of the pins to be used as ADC inputs.
2. Configuration of the ADC unit.
3. Configuration of the channels for use, including
   - Automatic clock setting using sampling time as an input
   - Automatic interrupt control
   - Sampling time control
4. Disabling the unit when no longer required and enabling low-power mode.
5. Control the ADC unit, 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.

2.27. Comparator A Driver

Comparator A compares a reference input voltage and an analog input voltage. The driver functions support the use of Comparator A1 and comparator A2 that share the voltage detection circuit with voltage monitor 1 and voltage monitor 2. Either "comparator A1 and comparator A2" or "voltage monitor 1 and voltage monitor 2" can be selected to use the voltage detection circuit. Providing the following operations:

1. Configuring two Comparator A, including:
   - Individual Reference and ComparisonInput Select
   - Comparison result monitor
   - Comparison result output
   - Interrupt or reset mode selection
   - Reset Negation selection
   - Non-maskable or maskable interrupt
   - Digital filter function and sample frequency
   - Compares whether the analog input voltage has passed through the reference input voltage by rising or falling.
2. Controlling two Comparator A
   - Enable / disable comparator A
3. Reading the status flag.

2.28. Data Operation Circuit Driver

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

1. Configuring and enabling the DOC.
2. Disabling the DOC.
3. Controlling operation including switching between comparison, addition and subtraction modes.
4. Writing data to the DOC.
5. Reading result from DOC

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
bool	stdbool.h	Boolean	0 (false) to 1 (true)
double	C	Floating point, 64 bits	±∞
uint8_t	stdint.h	Unsigned, 8 bits	0 to 255
uint16_t		Unsigned, 16 bits	0 to 2^{15} - 1
int32_t		Signed, 32 bits	-2^{31} to 2^{31} - 1
uint32_t		Unsigned, 32 bits	0 to 2^{32} - 1

3.2. General definitions

- PDL_NO_FUNC

Used as a parameter when there is no applicable function.

- PDL_NO_PTR

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

- PDL_NO_DATA

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

- PDL_MCU_GROUP

The MCU group supported by this build of the driver library. It is defined as RX220.

A usage example is:

```
#if PDL_MCU_GROUP != RX220
 #error "Wrong RPDL !"
#endif
```

- PDL_VERSION

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

A usage example is:

```
const uint16_t rpdl_version_number = PDL_VERSION;
```

3.2.1. 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 2 lists the Renesas Embedded APIs by peripheral function.

Table 2: Renesas Embedded API List

Category	Number	Name	Description
Clock Generation Circuit	1	R_CGC_Set	Configure the clock generation circuit.
	2	R_CGC_Control	Modify the clock generation circuit operation.
	3	R_CGC_GetStatus	Read the status of the clock generation circuit.
Interrupt control unit	1	R_INTC_SetExtInterrupt	Select the external interrupt pins.
	2	R_INTC_CreateExtInterrupt	Configure an external interrupt signal.
	3	R_INTC_CreateSoftwareInterrupt	Enable use of the software interrupt.
	4	R_INTC_CreateFastInterrupt	Assign handlers for the fixed-vector interrupts.
	5	R_INTC_CreateExceptionHandlers	Enable faster interrupt processing for one interrupt.
	6	R_INTC_ControlExtInterrupt	External interrupt control.
	7	R_INTC_GetExtInterruptStatus	Read the external interrupt status.
	8	R_INTC_Read	Read an interrupt register.
	9	R_INTC_Write	Update an interrupt register.
	10	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.
	9	R_IO_PORT_NotAvailable	Configure I/O port pins that are not available.
Multifunction Pin Controller	1	R_MPC_Read	Read a PFC register.
	2	R_MPC_Write	Write to a PFC register.
	3	R_MPC_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.
	3	R_MCU_OFS	Configure the device start-up operation.
Voltage Detection Circuit	1	R_LVD_Create	Configure the voltage detection circuit.
	2	R_LVD_Control	Control the voltage detection circuit.
	3	R_LVD_GetStatus	Check the status of the voltage detection module.
Clock Frequency Accuracy Measurement Circuit	1	R_CAC_Create	Configure the clock accuracy circuit.
	2	R_CAC_Destroy	Stop the clock accuracy circuit.
	3	R_CAC_Control	Control the clock accuracy circuit.
	4	R_CAC_GetStatus	Read the clock accuracy circuit status.
Low Power Consumption	1	R_LPC_Create	Configure the MCU low power conditions.
	2	R_LPC_Control	Select a low power consumption mode.
	5	R_LPC_GetStatus	Read the status flags.
Register Write Protection	1	R_RWP_Control	Control register write protection.
	2	R_RWP_GetStatus	Get the status of the register protection.
Bus Controller	1	R_BSC_Set	Configure the internal bus operation.
	2	R_BSC_Create	Configure the external bus controller.
	5	R_BSC_Control	Modify the External Bus Controller operation.
	7	R_BSC_GetStatus	Read the External Bus Controller status flags.
DMA Controller	1	R_DMAC_Create	Configure the DMA controller.
	2	R_DMAC_Destroy	Disable a DMA channel.
	3	R_DMAC_Control	Control the DMA controller.
	4	R_DMAC_GetStatus	Check the status of the DMA channel.

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.
Event Link Controller	1	R_ELC_Create	Enable the ELC module.
	2	R_ELC_Destroy	Disable the ELC module.
	3	R_ELC_Read	Read the ELC port buffer.
	4	R_ELC_Write	Write to the ELC port buffer.
	5	R_ELC_Control	Control the ELC.
Multi-function Timer pulse unit	1	R_MTU2_Set	Configure the Multi-function Timer Pulse Units.
	2	R_MTU2_Create	Configure a MTU channel.
	3	R_MTU2_Destroy	Disable a Multi-function Timer Pulse Unit.
	4	R_MTU2_ControlChannel	Control an MTU channel.
	5	R_MTU2_ControlUnit	Control a Multi-function Timer Pulse Unit.
	6	R_MTU2_ReadChannel	Read from MTU channel registers.
	7	R_MTU2_ReadUnit	Read from MTU 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 module status.
8-bit Timer	1	R_TMR_Set	Configure the optional TMR pins.
	2	R_TMR_CreateChannel	Configure a TMR timer channel.
	3	R_TMR_CreateUnit	Configure a TMR timer unit.
	4	R_TMR_CreatePeriodic	Select periodic operation.
	5	R_TMR_CreateOneShot	Configure and use a one-shot timer.
	6	R_TMR_Destroy	Disable a TMR timer unit.
	7	R_TMR_ControlChannel	Write to timer channel registers.
	8	R_TMR_ControlUnit	Write to timer unit registers.
	9	R_TMR_ControlPeriodic	Control periodic operation.
	10	R_TMR_ReadChannel	Read from timer channel registers.
	11	R_TMR_ReadUnit	Read from timer unit registers.
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.
Real-time Clock	1	R_RTC_Create	Configure the Real-time clock in calendar count mode.
	2	R_RTC_CreateBinary	Configure the RTC module in binary count mode.
	3	R_RTC_Destroy	Shut down the Real-time clock.
	4	R_RTC_Control	Modify the RTC clock operation in calendar count mode.
	5	R_RTC_ControlBinary	Modify the RTC module operation in binary count mode.
	6	R_RTC_Read	Read the Real-time clock status flags and counters in calendar count mode.
	7	R_RTC_ReadBinary	Read the RTC module status flags and counters in binary count mode.
	8	R_RTC_CreateWarm	Reconfigure RTC interrupt setting at warm start up in calendar count or mode binary count mode.
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_SPI_Transfer	Perform an SCI SPI transfer.
	7	R_SCI_IIC_Write	Perform an SCI IIC master write.
	8	R_SCI_IIC_Read	Perform an SCI IIC master read.
	9	R_SCI_IIC_ReadLastByte	Finish an SCI master read if using DMAC or DTC.
	10	R_SCI_Control	Control the SCI channel.
	11	R_SCI_GetStatus	Check the status of an SCI channel.

I²C bus interface	1	R_IIC_Set	Configure the I²C pin selection.
	2	R_IIC_Create	I²C channel setup.
	3	R_IIC_Destroy	Disable an I²C channel.
	4	R_IIC_MasterSend	Write data to a slave device.
	5	R_IIC_MasterReceive	Read data from a slave device.
	6	R_IIC_MasterReceiveLast	Complete a DMAC or DTC-based read process.
	7	R_IIC_SlaveMonitor	Monitor the bus and receive data from a master.
	8	R_IIC_SlaveSend	Write data to a master device.
	9	R_IIC_Control	I²C channel control.
	10	R_IIC_GetStatus	Read the status for an I²C channel.
Serial Peripheral Interface	1	R_SPI_Set	Configure the SPI pin selection.
	2	R_SPI_Create	Configure an SPI channel.
	3	R_SPI_Destroy	Shutdown an SPI channel.
	4	R_SPI_Command	Configure an SPI command.
	5	R_SPI_Transfer	Transfer data over an SPI channel.
	6	R_SPI_Control	Control an SPI channel.
	7	R_SPI_GetStatus	Check the status of an SPI 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.
12-bit Analog to Digital converter	1	R_ADC_12_Set	Select the I/O pins for the 12-bit ADC.
	2	R_ADC_12_CreateUnit	Configure the 12-bit ADC unit.
	3	R_ADC_12_CreateChannel	Configure 12-bit ADC analog channels.
	4	R_ADC_12_Destroy	Shut down the ADC unit.
	5	R_ADC_12_Control	Start or stop the ADC unit.
	6	R_ADC_12_Read	Read the ADC conversion results.
Comparator A	1	R_CPA_Create	Configure the Comparator A module.
	2	R_CPA_Control	Control the Comparator A module.
	3	R_CPA_GetStatus	Check the status of the Comparator A module.
Data Operation Circuit	1	R_DOC_Create	Configure the Data Operation Circuit.
	2	R_DOC_Destroy	Disable the Data Operation Circuit.
	3	R_DOC_Control	Control the Data Operation Circuit.
	4	R_DOC_Read	Read the Data Operation Circuit result.
	5	R_DOC_Write	Write data to the Data Operation Circuit.

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.

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

```
/* RPDL definitions */
#include "r_pdl_mpc.h"
#include "r_pdl_sci.h"

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

void func( void )
{
    bool result;

    /* Write 0xFF to register MPC1 */
    result = R_MPC_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 RPDL 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 RPDL API parameter list.

4.2.1. Clock Generation Circuit

1) R_CGC_Set

Synopsis

Configure the clock generation circuit.

Prototype

```
bool R_CGC_Set(
    uint8_t data1,      // Clock selection
    uint32_t data2,     // Configuration options
    double data3,       // Clock frequency
    double data4,       // System clock frequency
    double data5,       // Peripheral module clock D frequency
    double data6,       // Peripheral module clock B frequency
    double data7        // Flash interface clock frequency
    uint16_t data8      // Sub-clock stabilization time
);
```

Description (1/2)

Set a clock source frequencies and options.

[data1]
Clock source selection.

- Clock source selection

PDL_CGC_CLK_LOCO or PDL_CGC_CLK_HOCO or PDL_CGC_CLK_MAIN or PDL_CGC_CLK_SUB_CLOCK or PDL_CGC_CLK_IWDTLOCO	Select the low-speed on-chip oscillator (LOCO), high-speed on-chip oscillator (HOCO), main clock oscillator, sub-clock oscillator or IWDT-dedicated low-speed clock on-chip oscillator (IWDTLOCO).

[data2]
Configuration settings.

Options which are applicable only when the HOCO is selected in parameter data1.

- High-speed on-chip oscillator frequency selection

PDL_CGC_HOCO_32000 or PDL_CGC_HOCO_36864 or PDL_CGC_HOCO_40000 or PDL_CGC_HOCO_50000	Select the HOCO frequency (32.0, 36.864, 40.0 or 50.0 MHz).

Options which are applicable only when the Main clock oscillator is selected in parameter data1.

- Main clock oscillator type

PDL_CGC_MAIN_RESONATOR or PDL_CGC_MAIN_EXTERNAL	Select the oscillator type.

- Main clock oscillator drive type

PDL_CGC_MAIN_CERAMIC_LEAD_16_20	Adjust the drive level when a 16 to 20 MHz lead type ceramic resonator is used.

- Sub-clock oscillator drive ability (select only once when power on)
  The standard load capacitance is for the case the sub-clock resonator is not fitted.

PDL_CGC_SUB_CLOCK_CL_LOW or **PDL_CGC_SUB_CLOCK_CL_STANDARD**	Adjust the drive level for a crystal with low or standard load capacitance.

[data3]
The frequency of the selected clock source, in Hertz.

[data4]
The desired frequency of the System clock (ICLK), in Hertz.

[data5]
The desired frequency of the Peripheral module D clock (PCLKD), in Hertz.

[data6]
The desired frequency of the Peripheral module B clock (PCLKB), in Hertz.

Description (2/2)

[data7]
The desired frequency of the Flash memory interface clock (FCLK), in Hertz.

[data8]
Select the sub-clock oscillator stabilization times.
If no selections are required, specify PDL_NO_DATA.

- Sub-clock oscillator waiting time. Compulsory option if PDL_CGC_CLK_SUB_CLOCK is selected.(ignore for 48 pin package)

  Make sure to set the option whether Sub-clock is used or not based on the actual board situation

PDL_CGC_SUB_2 or PDL_CGC_SUB_4 or PDL_CGC_SUB_8 or PDL_CGC_SUB_16 or PDL_CGC_SUB_32 or PDL_CGC_SUB_64 or PDL_CGC_SUB_512 or PDL_CGC_SUB_1024 or PDL_CGC_SUB_2048 or PDL_CGC_SUB_4096 or PDL_CGC_SUB_16384 or PDL_CGC_SUB_32768 or PDL_CGC_SUB_65536 or PDL_CGC_SUB_131072 or PDL_CGC_SUB_262144 or PDL_CGC_SUB_524288	Select the oscillation settling time of the sub clock oscillator

Return value

True if all parameters are valid and exclusive; otherwise false.
For RX220, the following rules shall be checked:

- $f_{MAIN\_CLOCK\_OSCILLATOR} \leq 20$ MHz (≥ 1 MHz if a resonator is used).
- $f_{ICLK} \leq 32$ MHz
- $f_{PCLKB} \leq 32$ MHz
- $f_{PCLKD} \leq 32$ MHz
- $f_{FCLK} \leq 32$ MHz
- The frequencies of the internal clocks (ICLK, PCLKB, PCLKD and FCLK) are achievable: (selected clock source) ÷ 1, 2, 4, 8, 16, 32 or 64.

Category

Clock generation circuit

References

R_CGC_Control, R_MCU_GetStatus, R_LPC_Create

Remarks

- Call this function once to set the clock frequency for each clock source whether it is used as system clock or RTC count source.
- If the current clock source is selected in parameter data1, the frequencies of the internal clocks will be changed by this function.
  Because this can not be done if ROM/Flash Program Erase mode is set or if an operating power mode transition is in progress, this function will return false if this is detected.
- After a power-on reset, the MCU selects the LOCO as the clock source.
- This function must be called before configuring clock-dependent modules.
- This function will enable the selected clock but will not select it as the current clock source. After the required settling time, use R_CGC_Control to select the desired clock source.
- If the sub-clock oscillator is not fitted, use R_CGC_Control to disable the oscillation circuit.
- The registers MOSCWTCR (main clock) and SOSCWTCR (sub-clock) provide stabilisation delays for the respective oscillator and must be written to while that clock is stopped. If any of these registers needs to be modified, stop the clock (using R_CGC_Control) and call R_LPC_Create to set the new value.
- If the IWDTLOCO is selected, specify PDL_NO_DATA for parameters data2 and data4 to data7.
- If the HOCO is selected, the HOCO power must not be turned off.
- If the sub-clock will be selected while in low-speed operating mode 2 (see R_LPC_Create), $f_{ICLK(Sub\text{-}clock)}$ and $f_{FCLK(Sub\text{-}clock)}$ must equal $f_{SUB\text{-}CLOCK}$.
- Make sure PCLKB clock frequency ≥ RTC count source clock frequency.
- When RTC is not to be used, call R_CGC_Control with option PDL_CGC_RTC_NOT_USE after calling this function to configure the RTC count source.
- Sub-clock oscillator is not available for 48 pin package.

Program example

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

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

void func(void)
{
    /* Configure main clock operation using a 20.0 MHz crystal */
    /* ICLK = 20 MHz, PCLKD = 20 MHz, PCLKB = 20 MHz , FCLK = 20 MHz */
    R_CGC_Set(
        PDL_CGC_CLK_MAIN,
        PDL_CGC_MAIN_RESONATOR,
        20E6,
        20E6,
        20E6,
        20E6,
        20E6,
        PDL_NO_DATA
    );

    /* Configure HOCO operation. */
    /* ICLK = 32 MHz, PCLKD = 32 MHz, PCLKB = 32 MHz , FCLK = 32 MHz */
R_CGC_Set(
        PDL_CGC_CLK_HOCO
        PDL_CGC_HOCO_32000,
        32E6,
        32E6,
        32E6,
        32E6,
        32E6,
        PDL_NO_DATA
    );
}
```

2) R_CGC_Control

Synopsis

Modify the clock generation circuit operation.

Prototype

```
bool R_CGC_Control(
    uint8_t data1,     // Clock selection
    uint32_t data2,    // Clock control options
    uint8_t data3      // Clock control options
);
```

Description (1/2)

Modify the clock control registers.

[data1]
Clock source selection. If no change is required, specify PDL_NO_DATA.

- Clock source selection

PDL_CGC_CLK_LOCO or PDL_CGC_CLK_HOCO or PDL_CGC_CLK_MAIN or PDL_CGC_CLK_SUB_CLOCK	Select the low-speed on-chip oscillator (LOCO), high-speed on-chip oscillator (HOCO), main clock oscillator or sub-clock oscillator

[data2]
Clock control selection.
All selections are optional. If no change is required, specify PDL_NO_DATA.
If multiple selections are required, use "|" to separate each selection.

- Low-speed on-chip oscillator control

PDL_CGC_LOCO_ENABLE or PDL_CGC_LOCO_DISABLE	Enable or disable the LOCO.

- High-speed on-chip oscillator control

PDL_CGC_HOCO_ENABLE or PDL_CGC_HOCO_DISABLE	Enable or disable the HOCO.

- High-speed on-chip oscillator power control

PDL_CGC_HOCO_POWER_ON or PDL_CGC_HOCO_POWER_OFF	Control the HOCO power supply.

- Main clock oscillator control

PDL_CGC_MAIN_ENABLE or PDL_CGC_MAIN_DISABLE	Enable or disable the main clock oscillator.

- Main clock Oscillation Stop Detection control

PDL_CGC_OSC_STOP_ENABLE or PDL_CGC_OSC_STOP_INTERRUPT or PDL_CGC_OSC_STOP_DISABLE	Enable (without or with interrupt request output) or disable the oscillation stop detection function for the main clock oscillator.

- Main clock Oscillation Stop Detection flag control

PDL_CGC_OSC_STOP_CLEAR_FLAG	Clear the main clock oscillation stop detection flag.

[data3]
Clock control selection.
All selections are optional. If no change is required, specify PDL_NO_DATA.
If multiple selections are required, use "|" to separate each selection.

- Sub-clock oscillator control (Ignore for 48 pin package)

PDL_CGC_SUB_CLOCK_ENABLE or PDL_CGC_SUB_CLOCK_DISABLE	Enable or disable the sub-clock oscillator.

- IWDT-dedicated low-speed on-chip oscillator control

PDL_CGC_IWDTLOCO_ENABLE or PDL_CGC_IWDTLOCO_DISABLE	Enable or disable the IWDTLOCO.

Description (2/2)

- RTC initialization control

PDL_CGC_RTC_TO_BE_USED or PDL_CGC_RTC_NOT_USE	Select whether RTC will be used.

Return value

True if all parameters are valid and exclusive and a selected clock source has been configured; otherwise false.

Category

Clock generation circuit

References

R_CGC_Set, R_LPC_GetStatus, R_LPC_Create

Remarks

- Use R_CGC_Set to configure a clock source before calling this function.
- While the main clock Oscillation Stop Detection feature is enabled, the LOCO is started and cannot be stopped.
- Clearing the main clock Oscillation Stop Detection flag will not succeed until a clock source other than the main oscillator is selected (using parameter data1).
- If the main clock Oscillation Stop Detection flag is cleared, the interrupt output is also disabled. Use this function to re-enable the interrupt output after the main clock oscillation has been restored.
- Do not stop a clock that is in use.
- Do not change the clock source if an Operating Power Control Mode transition is taking place (see R_LPC_GetStatus).
- If low-speed operating mode 2 is selected (see R_LPC_Create), disable the HOCO.
- If middle-speed operating modes 1 or 2 are selected (seeR_LPC_Create), do not change the HOCO power state.
- If the main clock oscillator pins will be used as general I/O, call this function with PDL_CGC_MAIN_DISABLE and PDL_CGC_MAIN_FORCED_DISABLE selected in parameter data2.
- If this function is used to enable a clock oscillator, wait for the required settling time before selecting the clock source.
- This function can not be used if ROM/Flash Program Erase mode is set or if an operating power mode transition is in progress. This function will return false if this is detected.
- Calling R_RTC_Create after using option PDL_CGC_RTC_TO_BE_USED
- Make sure PCLKB clock frequency ≥ RTC count source clock frequency.
- Call R_CGC_Set once to set sub- clock frequency before call R_CGC_Control with option PDL_CGC_RTC_TO_BE_USE.
- Sub-clock oscillator is not available for 48 pin package.

Program example

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

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

void func(void)
{
    /* Stop the sub-clock oscillator */
    R_CGC_Control(
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_CGC_SUB_CLOCK_DISABLE
    );

    /* Select the Main clock */
    R_CGC_Control(
        PDL_CGC_CLK_MAIN,
        PDL_NO_DATA,
        PDL_NO_DATA
    );
}
```

3) R_CGC_GetStatus

Synopsis

Read the status of the clock generation circuit.

Prototype

```
bool R_CGC_GetStatus(
    uint16_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.

b15	b14	b13	b12	b11	b10	b9	b8
0	HOCO power	Clock control					
		HOCO	IWDTLOCO	LOCO	Sub-clock	Main clock	-
	0: Power on 1: Power off	0: Operating 1: Stopped					

b7	b6 - b4	b3 - b2	b1	b0
0	Selected clock source	0	Main clock oscillation stop detection	
	000b: LOCO 001b: HOCO 010b: Main clock 011b: Sub-clock		0: Disabled 1: Enabled	0: Normal operation 1: Stop detected

Return value

True.

Category

Clock generation circuit

References

R_CGC_Control

Remarks

- Use R_CGC_Control to clear the main clock oscillation stop detection flag.

Program example

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

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

void func(void)
{
    uint16_t Status_flags;

    R_CGC_GetStatus(
        &Status_flags
    );
}
```

4.2.2. Interrupt Control Unit

1) R_INTC_SetExtInterrupt

Synopsis

Select the external interrupt pins.

Prototype

```
bool R_INTC_SetExtInterrupt(
    uint32_t data      // Pin selection
);
```

Description (1/2)

Assign the external interrupt pins.

[data]
Allocate the pins for signals IRQ0 to IRQ7. All selections are optional.
If multiple selections are required, use "|" to separate each selection.
If no pins are required, specify PDL_NO_DATA.

PDL_INTC_IRQ0_P30 or PDL_INTC_IRQ0_P10 or PDL_INTC_IRQ0_PD0	Select the pins to be used for signals IRQ0 to IRQ7.
PDL_INTC_IRQ1_P31 or PDL_INTC_IRQ1_P11 or PDL_INTC_IRQ1_PD1	
PDL_INTC_IRQ2_P32 or PDL_INTC_IRQ2_P12 or PDL_INTC_IRQ2_PD2	
PDL_INTC_IRQ3_P33 or PDL_INTC_IRQ3_P13 or PDL_INTC_IRQ3_PD3	
PDL_INTC_IRQ4_PB1 or PDL_INTC_IRQ4_P14 or PDL_INTC_IRQ4_P34 or PDL_INTC_IRQ4_PD4	
PDL_INTC_IRQ5_PA4 or PDL_INTC_IRQ5_P15 or PDL_INTC_IRQ5_PD5 or PDL_INTC_IRQ5_PE5	
PDL_INTC_IRQ6_PA3 or PDL_INTC_IRQ6_P16 or PDL_INTC_IRQ6_PD6 or PDL_INTC_IRQ6_PE6	
PDL_INTC_IRQ7_PE2 or PDL_INTC_IRQ7_P17 or PDL_INTC_IRQ7_PD7 or PDL_INTC_IRQ7_PE7	

Return value

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

Category

Interrupt control

References

R_INTC_CreateExtInterrupt

Remarks

- Before calling R_INTC_CreateExtInterrupt, call this function to select the required pins.
- The Multifunction Pin Control registers are modified to enable each selected IRQ pin and the I/O Port PMR and PDR registers are modified to set the pin as an input.
- A pin can be used both as an interrupt input and a peripheral or general purpose input or output (apart from an analog input). If the dual operation is required, call this function before configuring the peripheral or I/O port operation.
- Some pin options are not available on smaller device packages.

Program example

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

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

void func(void)
{
    /* Select P30 for IRQ0 and P31 for IRQ1 */
    R_INTC_SetExtInterrupt(
        PDL_INTC_IRQ0_P30 | PDL_INTC_IRQ1_P31
    );
}
```

2) R_INTC_CreateExtInterrupt

Synopsis

Configure an external interrupt signal.

Prototype

```
bool R_INTC_CreateExtInterrupt(
    uint8_t data1,      // Signal selection
    uint32_t data2,     // Configuration
    void * func,        // Callback function
    uint8_t data3       // Interrupt priority level
);
```

Description (1/2)

Sets the specified interrupt detection and control.

[data1]
Choose the interrupt signal to be configured.

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

[data2]
Choose the settings. If multiple selections are required, use "|" to separate each selection.
The default settings are shown in **bold**.

- Digital filter selection

PDL_INTC_FILTER_DISABLE or PDL_INTC_FILTER_DIV_1 or PDL_INTC_FILTER_DIV_8 or PDL_INTC_FILTER_DIV_32 or PDL_INTC_FILTER_DIV_64	The interrupt pin input can be unfiltered or sampled using the peripheral clock PCLKB divided by 1, 8, 32 or 64. For the NMI signal, this selection is ignored if the NMI pin is not enabled.

Options which only apply to the IRQ pins

- Input sense selection

PDL_INTC_LOW 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.

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

PDL_INTC_DMAC_DTC_TRIGGER_DISABLE or PDL_INTC_DMAC_TRIGGER_ENABLE or PDL_INTC_DTC_TRIGGER_ENABLE	Disable or enable activation of the DMAC or DTC when a valid edge transition is detected on a valid IRQn pin.

Options which only apply to the NMI

- Pin enable and input sense selection

PDL_INTC_FALLING or PDL_INTC_RISING	Enable the NMI pin and select falling or rising edge detection. Required only if the NMI pin is to be used.

- Internal detection control

PDL_INTC_OSD_DISABLE or PDL_INTC_OSD_ENABLE	Disable or enable the NMI signal when the oscillation stop detection interrupt occurs.
PDL_INTC_IWDT_DISABLE or PDL_INTC_IWDT_ENABLE	Disable or enable the NMI signal when an IWDT underflow interrupt occurs.
PDL_INTC_LVD1_DISABLE or PDL_INTC_LVD1_ENABLE	Disable or enable the NMI signal when a low-voltage detection 1 interrupt occurs.
PDL_INTC_LVD2_DISABLE or PDL_INTC_LVD2_ENABLE	Disable or enable the NMI signal when a low-voltage detection 2 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.

Description (2/2)

[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 and is ignored.

Return value

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

Category

Interrupt control

Reference

R_INTC_SetExtInterrupt

Remarks

- Function R_INTC_SetExtInterrupt must be called before any use of this function.
- The selected interrupt is enabled automatically.
- Please see the notes on callback function use in §6.
- The NMI callback function should not return. It should stop operation or reset the system.
- If the NMI interrupt fails to initialise, this function will return false.

Program example

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

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

/* Callback function */
void CallBackFunc( void ){}

void func( void )
{
    /* Configure the IRQ1 interrupt */
    R_INTC_CreateExtInterrupt(
        PDL_INTC_IRQ1,
        PDL_INTC_FALLING,
        CallBackFunc,
        7
    );

    /* Configure the NMI pin */
    R_INTC_CreateExtInterrupt(
        PDL_INTC_NMI,
        PDL_INTC_FALLING,
        CallBackFunc,
        15
    );

    /* Configure the NMI triggered by the IWDT only (no NMI pin) */
    R_INTC_CreateExtInterrupt(
        PDL_INTC_NMI,
        PDL_INTC_IWDT_ENABLE,
        CallBackFunc,
        10
    );
}
```

3) 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.

PDL_INTC_DTC_SW_TRIGGER_DISABLE 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
    );
}
```

4) 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_FERRF	Clock frequency accuracy measurement	Frequency error
PDL_INTC_VECTOR_MENDF		Measurement end
PDL_INTC_VECTOR_OVFF		Overflow
PDL_INTC_VECTOR_SPEI0	SPI 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_DOPCF	Data operation	Condition detection
PDL_INTC_VECTOR_CUP	Real-time clock	Carry
PDL_INTC_VECTOR_ALM		Alarm
PDL_INTC_VECTOR_PRD		Periodic
PDL_INTC_VECTOR_IRQ0	Interrupt controller	Valid edge or level detected on an external interrupt pin
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_LVD1	Low voltage detection	Voltage detection
PDL_INTC_VECTOR_LVD2		Voltage detection
PDL_INTC_VECTOR_CMPA1	Comparator A	Voltage detection
PDL_INTC_VECTOR_CMPA2		Voltage detection
PDL_INTC_VECTOR_S12ADI0	12-bit ADC	Conversion completed
PDL_INTC_VECTOR_GBADI		Group B scan completed
PDL_INTC_VECTOR_ELSR18I	Event link controller	Event interrupt
PDL_INTC_VECTOR_TGIA0	Multi-function Timer Pulse Unit, channel 0	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

Description (2/2)

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
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_OEI1	Port Output Enable	Input-level sampling or output-level comparison detection
PDL_INTC_VECTOR_OEI2		
PDL_INTC_VECTOR_CMIA0	8-bit timer TMR channel 0	Compare match A
PDL_INTC_VECTOR_CMIB0		Compare match B
PDL_INTC_VECTOR_OVI0		Overflow
PDL_INTC_VECTOR_CMIA1	8-bit timer TMR channel 1	Compare match A
PDL_INTC_VECTOR_CMIB1		Compare match B
PDL_INTC_VECTOR_OVI1		Overflow
PDL_INTC_VECTOR_CMIA2	8-bit timer TMR channel 2	Compare match A
PDL_INTC_VECTOR_CMIB2		Compare match B
PDL_INTC_VECTOR_OVI2		Overflow
PDL_INTC_VECTOR_CMIA3	8-bit timer TMR channel 3	Compare match A
PDL_INTC_VECTOR_CMIB3		Compare match B
PDL_INTC_VECTOR_OVI3		Overflow
PDL_INTC_VECTOR_DMAC0I	Direct memory access controller	Transfer complete or Transfer escape end
PDL_INTC_VECTOR_DMAC1I		
PDL_INTC_VECTOR_DMAC2I		
PDL_INTC_VECTOR_DMAC3I		
PDL_INTC_VECTOR_EXDMAC0I	External DMAC	Transfer complete or Transfer escape end
PDL_INTC_VECTOR_EXDMAC1I		
PDL_INTC_VECTOR_ERI0	SCI channel 0	Error in data received
PDL_INTC_VECTOR_RXI0		Data received
PDL_INTC_VECTOR_TXI0		Start of next data transfer
PDL_INTC_VECTOR_TEI0		End of data transfer
PDL_INTC_VECTOR_ERI5	SCI channel 5	Error in data received
PDL_INTC_VECTOR_RXI5		Data received
PDL_INTC_VECTOR_TXI5		Start of next data transfer
PDL_INTC_VECTOR_TEI5		End of data transfer
PDL_INTC_VECTOR_ERI6	SCI channel 6	Error in data received
PDL_INTC_VECTOR_RXI6		Data received
PDL_INTC_VECTOR_TXI6		Start of next data transfer
PDL_INTC_VECTOR_TEI6		End of data transfer
PDL_INTC_VECTOR_ERI9	SCI channel 9	Error in data received
PDL_INTC_VECTOR_RXI9		Data received
PDL_INTC_VECTOR_TXI9		Start of next data transfer
PDL_INTC_VECTOR_TEI9		End of data transfer
PDL_INTC_VECTOR_ERI12	SCI channel 12	Error in data received
PDL_INTC_VECTOR_RXI12		Data received
PDL_INTC_VECTOR_TXI12		Start of next data transfer
PDL_INTC_VECTOR_TEI12		End of data transfer
PDL_INTC_VECTOR_SCIX0		Extended serial mode, Break field
PDL_INTC_VECTOR_SCIX1		Extended serial mode, Control field
PDL_INTC_VECTOR_SCIX2		Extended serial mode, Bus collision
PDL_INTC_VECTOR_SCIX3		Extended serial mode, Valid edge
PDL_INTC_VECTOR_ICEEI0	I²C bus interface channel 0	Transfer error or event generation
PDL_INTC_VECTOR_ICRXI0		Data received
PDL_INTC_VECTOR_ICTXI0		Start of next data transfer
PDL_INTC_VECTOR_ICTEI0		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_IRQ2
  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) */
```

5) R_INTC_CreateExceptionHandlers

Synopsis

Assign handlers for the fixed-vector interrupts.

Prototype

```
bool R_INTC_CreateExceptionHandlers(
    void * func1,     // Callback function
    void * func2      // 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.

Return value

True.

Category

Interrupt control

Reference

None

Remarks

- Please see the notes on callback function use in §6.

Program example

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

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

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

void func( void )
{
    /* Assign a function to manage undefined instruction errors */
    R_INTC_CreateExceptionHandlers(
        PDL_NO_FUNC,
        UndefinedInstruction
    );
}
```

6) R_INTC_ControlExtInterrupt

Synopsis

External interrupt control.

Prototype

```
bool R_INTC_ControlExtInterrupt(
    uint8_t data1,      // Pin selection
    uint32_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.

- Digital filter selection

PDL_INTC_FILTER_DISABLE or PDL_INTC_FILTER_DIV_1 or PDL_INTC_FILTER_DIV_8 or PDL_INTC_FILTER_DIV_32 or PDL_INTC_FILTER_DIV_64	Disable the filter or select PCLKB divided by 1, 8, 32 or 64.

- Detection sense selection (for the IRQ pins)

PDL_INTC_LOW 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.

- Interrupt request clearing

PDL_INTC_CLEAR_IR_FLAG	Clear the IRQ or NMI interrupt request flag. This is not required if: • A callback function has been specified. • The interrupt priority level is higher than 0. • The processor interrupt priority level is lower than the interrupt priority level. This operation should not be applied when low-level detection is used.
PDL_INTC_CLEAR_OSD_FLAG	Clear the Oscillation Stop detection NMI flag.
PDL_INTC_CLEAR_IWDT_FLAG	Clear the IWDT event detection NMI flag.
PDL_INTC_CLEAR_LVD1_FLAG	Clear the LVD1 event detection NMI flag.
PDL_INTC_CLEAR_LVD2_FLAG	Clear the LVD2 event detection NMI 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
    );
}
```

7) 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
	Detection condition	Current level	Status
0	00: Low level 01: Falling edge 10: Rising edge 11: Both edges	0: Low 1: High	0: Not detected 1: Detected

For the NMI interrupt:

b7	b6	b5	b4	b3	b2	b1	b0
Other interrupt request					NMI pin		
LVD2	LVD1	Underflow		Oscillation stop	Current level	Detection condition	Request status
		IWDT	-				
0: Not detected 1: Detected					0: Low 1: High	0: Falling 1: Rising	0: Not detected 1: Detected

Return value

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

Category

Interrupt control

Reference

R_INTC_ControlExtInterrupt

Remarks

- The MPC registers are used 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.
- Clear the NMI status flags using R_INTC_ControlExtInterrupt.

Program example

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

/* RPDL 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_MTU4_TGIA
PDL_INTC_REG_IR_FCU_FIFERR	PDL_INTC_REG_IR_MTU4_TGIB
PDL_INTC_REG_IR_FCU_FRDYI	PDL_INTC_REG_IR_MTU4_TGIC
PDL_INTC_REG_IR_ICU_SWINT	PDL_INTC_REG_IR_MTU4_TGID
PDL_INTC_REG_IR_CMT0_CMI	PDL_INTC_REG_IR_MTU4_TCIV
PDL_INTC_REG_IR_CMT1_CMI	PDL_INTC_REG_IR_MTU5_TGIU
PDL_INTC_REG_IR_CMT2_CMI	PDL_INTC_REG_IR_MTU5_TGIV
PDL_INTC_REG_IR_CMT3_CMI	PDL_INTC_REG_IR_MTU5_TGIW
PDL_INTC_REG_IR_CAC_FERRF	PDL_INTC_REG_IR_POE_OEI1
PDL_INTC_REG_IR_CAC_MENDF	PDL_INTC_REG_IR_POE_OEI2
PDL_INTC_REG_IR_CAC_OVFF	PDL_INTC_REG_IR_TMR0_CMIA
PDL_INTC_REG_IR_SPI0_SPEI	PDL_INTC_REG_IR_TMR0_CMIB
PDL_INTC_REG_IR_SPI0_SPRI	PDL_INTC_REG_IR_TMR0_OVI
PDL_INTC_REG_IR_SPI0_SPTI	PDL_INTC_REG_IR_TMR1_CMIA
PDL_INTC_REG_IR_SPI0_SPII	PDL_INTC_REG_IR_TMR1_CMIB
PDL_INTC_REG_IR_DOC_DOPCF	PDL_INTC_REG_IR_TMR1_OVI
PDL_INTC_REG_IR_RTC_CUP	PDL_INTC_REG_IR_TMR2_CMIA
PDL_INTC_REG_IR_ICU_IRQ0	PDL_INTC_REG_IR_TMR2_CMIB
PDL_INTC_REG_IR_ICU_IRQ1	PDL_INTC_REG_IR_TMR2_OVI
PDL_INTC_REG_IR_ICU_IRQ2	PDL_INTC_REG_IR_TMR3_CMIA
PDL_INTC_REG_IR_ICU_IRQ3	PDL_INTC_REG_IR_TMR3_CMIB
PDL_INTC_REG_IR_ICU_IRQ4	PDL_INTC_REG_IR_TMR3_OVI
PDL_INTC_REG_IR_ICU_IRQ5	PDL_INTC_REG_IR_DMAC_DMAC0I
PDL_INTC_REG_IR_ICU_IRQ6	PDL_INTC_REG_IR_DMAC_DMAC1I
PDL_INTC_REG_IR_ICU_IRQ7	PDL_INTC_REG_IR_DMAC_DMAC2I
PDL_INTC_REG_IR_LVD_LVD1	PDL_INTC_REG_IR_DMAC_DMAC3I
PDL_INTC_REG_IR_LVD_LVD2	PDL_INTC_REG_IR_SCI1_ERI
PDL_INTC_REG_IR_CMPA_CMPA1	PDL_INTC_REG_IR_SCI1_RXI
PDL_INTC_REG_IR_CMPA_CMPA2	PDL_INTC_REG_IR_SCI1_TXI
PDL_INTC_REG_IR_RTC_ALM	PDL_INTC_REG_IR_SCI1_TEI
PDL_INTC_REG_IR_RTC_PRD	PDL_INTC_REG_IR_SCI5_ERI
PDL_INTC_REG_IR_S12AD_S12ADI	PDL_INTC_REG_IR_SCI5_RXI
PDL_INTC_REG_IR_S12AD_GBADI	PDL_INTC_REG_IR_SCI5_TXI
PDL_INTC_REG_IR_ELC_ELSR18I	PDL_INTC_REG_IR_SCI5_TEI
PDL_INTC_REG_IR_MTU0_TGIA	PDL_INTC_REG_IR_SCI6_ERI
PDL_INTC_REG_IR_MTU0_TGIB	PDL_INTC_REG_IR_SCI6_RXI
PDL_INTC_REG_IR_MTU0_TGIC	PDL_INTC_REG_IR_SCI6_TXI
PDL_INTC_REG_IR_MTU0_TGID	PDL_INTC_REG_IR_SCI6_TEI
PDL_INTC_REG_IR_MTU0_TCIV	PDL_INTC_REG_IR_SCI9_ERI
PDL_INTC_REG_IR_MTU0_TGIE	PDL_INTC_REG_IR_SCI9_RXI
PDL_INTC_REG_IR_MTU0_TGIF	PDL_INTC_REG_IR_SCI9_TXI
PDL_INTC_REG_IR_MTU1_TGIA	PDL_INTC_REG_IR_SCI9_TEI
PDL_INTC_REG_IR_MTU1_TGIB	PDL_INTC_REG_IR_SCI12_ERI
PDL_INTC_REG_IR_MTU1_TCIV	PDL_INTC_REG_IR_SCI12_RXI
PDL_INTC_REG_IR_MTU1_TCIU	PDL_INTC_REG_IR_SCI12_TXI
PDL_INTC_REG_IR_MTU2_TGIA	PDL_INTC_REG_IR_SCI12_TEI
PDL_INTC_REG_IR_MTU2_TGIB	PDL_INTC_REG_IR_SCI12_SCIX0
PDL_INTC_REG_IR_MTU2_TCIV	PDL_INTC_REG_IR_SCI12_SCIX1
PDL_INTC_REG_IR_MTU2_TCIU	PDL_INTC_REG_IR_SCI12_SCIX2
PDL_INTC_REG_IR_MTU3_TGIA	PDL_INTC_REG_IR_SCI12_SCIX3
PDL_INTC_REG_IR_MTU3_TGIB	PDL_INTC_REG_IR_IIC0_EEI
PDL_INTC_REG_IR_MTU3_TGIC	PDL_INTC_REG_IR_IIC0_RXI
PDL_INTC_REG_IR_MTU3_TGID	PDL_INTC_REG_IR_IIC0_TXI
PDL_INTC_REG_IR_MTU3_TCIV	PDL_INTC_REG_IR_IIC0_TEI

IER register definitions

PDL_INTC_REG_IER02	PDL_INTC_REG_IER10
PDL_INTC_REG_IER03	PDL_INTC_REG_IER11
PDL_INTC_REG_IER04	PDL_INTC_REG_IER15
PDL_INTC_REG_IER05	PDL_INTC_REG_IER16
PDL_INTC_REG_IER07	PDL_INTC_REG_IER17
PDL_INTC_REG_IER08	PDL_INTC_REG_IER18
PDL_INTC_REG_IER0B	PDL_INTC_REG_IER19
PDL_INTC_REG_IER0C	PDL_INTC_REG_IER1B
PDL_INTC_REG_IER0D	PDL_INTC_REG_IER1C
PDL_INTC_REG_IER0E	PDL_INTC_REG_IER1D
PDL_INTC_REG_IER0F	PDL_INTC_REG_IER1E
	PDL_INTC_REG_IER1F

IPR register definitions

PDL_INTC_REG_IPR_BSC_BUSERR	PDL_INTC_REG_IPR_MTU1_TGIAB
PDL_INTC_REG_IPR_FCU_FIFERR	PDL_INTC_REG_IPR_MTU1_TCIVU
PDL_INTC_REG_IPR_FCU_FRDYI	PDL_INTC_REG_IPR_MTU2_TGIAB
PDL_INTC_REG_IPR_ICU_SWINT	PDL_INTC_REG_IPR_MTU2_TCIVU
PDL_INTC_REG_IPR_CMT0_CMI	PDL_INTC_REG_IPR_MTU3_TGIAD
PDL_INTC_REG_IPR_CMT1_CMI	PDL_INTC_REG_IPR_MTU3_TCIV
PDL_INTC_REG_IPR_CMT2_CMI	PDL_INTC_REG_IPR_MTU4_TGIAD
PDL_INTC_REG_IPR_CMT3_CMI	PDL_INTC_REG_IPR_MTU4_TCIV
PDL_INTC_REG_IPR_CAC_FERRF	PDL_INTC_REG_IPR_MTU5_TGI
PDL_INTC_REG_IPR_CAC_MENDF	PDL_INTC_REG_IPR_POE_OEI1
PDL_INTC_REG_IPR_CAC_OVFF	PDL_INTC_REG_IPR_POE_OEI2
PDL_INTC_REG_IPR_SPI0	PDL_INTC_REG_IPR_TMR0
PDL_INTC_REG_IPR_DOC_DOPCF	PDL_INTC_REG_IPR_TMR1
PDL_INTC_REG_IPR_RTC_CUP	PDL_INTC_REG_IPR_TMR2
PDL_INTC_REG_IPR_ICU_IRQ0	PDL_INTC_REG_IPR_TMR3
PDL_INTC_REG_IPR_ICU_IRQ1	PDL_INTC_REG_IPR_DMAC_DMAC0I
PDL_INTC_REG_IPR_ICU_IRQ2	PDL_INTC_REG_IPR_DMAC_DMAC1I
PDL_INTC_REG_IPR_ICU_IRQ3	PDL_INTC_REG_IPR_DMAC_DMAC2I
PDL_INTC_REG_IPR_ICU_IRQ4	PDL_INTC_REG_IPR_DMAC_DMAC3I
PDL_INTC_REG_IPR_ICU_IRQ5	PDL_INTC_REG_IPR_SCI1
PDL_INTC_REG_IPR_ICU_IRQ6	PDL_INTC_REG_IPR_SCI5
PDL_INTC_REG_IPR_ICU_IRQ7	PDL_INTC_REG_IPR_SCI6
PDL_INTC_REG_IPR_LVD_LVD1	PDL_INTC_REG_IPR_SCI9
PDL_INTC_REG_IPR_LVD_LVD2	PDL_INTC_REG_IPR_SCI12
PDL_INTC_REG_IPR_CMPA_CMPA1	PDL_INTC_REG_IPR_SCI12_SCIX0
PDL_INTC_REG_IPR_CMPA_CMPA2	PDL_INTC_REG_IPR_SCI12_SCIX1
PDL_INTC_REG_IPR_RTC_ALM	PDL_INTC_REG_IPR_SCI12_SCIX2
PDL_INTC_REG_IPR_RTC_PRD	PDL_INTC_REG_IPR_SCI12_SCIX3
PDL_INTC_REG_IPR_S12AD_S12ADI	PDL_INTC_REG_IPR_IIC0_EEI
PDL_INTC_REG_IPR_S12AD_GBADI	PDL_INTC_REG_IPR_IIC0_RXI
PDL_INTC_REG_IPR_ELC_ELSR18I	PDL_INTC_REG_IPR_IIC0_TXI
PDL_INTC_REG_IPR_MTU0_TGIAD	PDL_INTC_REG_IPR_IIC0_TEI
PDL_INTC_REG_IPR_MTU0_TCIVF	

DTCER register definitions

PDL_INTC_REG_DTCER_ICU_SWINT	PDL_INTC_REG_DTCER_MTU4_TGIA
PDL_INTC_REG_DTCER_CMT0_CMI	PDL_INTC_REG_DTCER_MTU4_TGIB
PDL_INTC_REG_DTCER_CMT1_CMI	PDL_INTC_REG_DTCER_MTU4_TGIC
PDL_INTC_REG_DTCER_CMT2_CMI	PDL_INTC_REG_DTCER_MTU4_TGID
PDL_INTC_REG_DTCER_CMT3_CMI	PDL_INTC_REG_DTCER_MTU4_TCIV
PDL_INTC_REG_DTCER_SPI0_SPRI	PDL_INTC_REG_DTCER_MTU5_TGIU
PDL_INTC_REG_DTCER_SPI0_SPTI	PDL_INTC_REG_DTCER_MTU5_TGIV
PDL_INTC_REG_DTCER_ICU_IRQ0	PDL_INTC_REG_DTCER_MTU5_TGIW
PDL_INTC_REG_DTCER_ICU_IRQ1	PDL_INTC_REG_DTCER_TMR0_CMIA
PDL_INTC_REG_DTCER_ICU_IRQ2	PDL_INTC_REG_DTCER_TMR0_CMIB
PDL_INTC_REG_DTCER_ICU_IRQ3	PDL_INTC_REG_DTCER_TMR1_CMIA
PDL_INTC_REG_DTCER_ICU_IRQ4	PDL_INTC_REG_DTCER_TMR1_CMIB
PDL_INTC_REG_DTCER_ICU_IRQ5	PDL_INTC_REG_DTCER_TMR2_CMIA
PDL_INTC_REG_DTCER_ICU_IRQ6	PDL_INTC_REG_DTCER_TMR2_CMIB
PDL_INTC_REG_DTCER_ICU_IRQ7	PDL_INTC_REG_DTCER_TMR3_CMIA
PDL_INTC_REG_DTCER_S12AD_S12ADI	PDL_INTC_REG_DTCER_TMR3_CMIB
PDL_INTC_REG_DTCER_S12AD_GBADI	PDL_INTC_REG_DTCER_DMAC_DMAC0I
PDL_INTC_REG_DTCER_ELC_ELSR18I	PDL_INTC_REG_DTCER_DMAC_DMAC1I
PDL_INTC_REG_DTCER_MTU0_TGIA	PDL_INTC_REG_DTCER_DMAC_DMAC2I
PDL_INTC_REG_DTCER_MTU0_TGIB	PDL_INTC_REG_DTCER_DMAC_DMAC3I
PDL_INTC_REG_DTCER_MTU0_TGIC	PDL_INTC_REG_DTCER_SCI1_RXI
PDL_INTC_REG_DTCER_MTU0_TGID	PDL_INTC_REG_DTCER_SCI1_TXI
PDL_INTC_REG_DTCER_MTU1_TGIA	PDL_INTC_REG_DTCER_SCI5_RXI
PDL_INTC_REG_DTCER_MTU1_TGIB	PDL_INTC_REG_DTCER_SCI5_TXI
PDL_INTC_REG_DTCER_MTU2_TGIA	PDL_INTC_REG_DTCER_SCI6_RXI
PDL_INTC_REG_DTCER_MTU2_TGIB	PDL_INTC_REG_DTCER_SCI6_TXI
PDL_INTC_REG_DTCER_MTU3_TGIA	PDL_INTC_REG_DTCER_SCI9_RXI
PDL_INTC_REG_DTCER_MTU3_TGIB	PDL_INTC_REG_DTCER_SCI9_TXI
PDL_INTC_REG_DTCER_MTU3_TGIC	PDL_INTC_REG_DTCER_SCI12_RXI
PDL_INTC_REG_DTCER_MTU3_TGID	PDL_INTC_REG_DTCER_SCI12_TXI
	PDL_INTC_REG_DTCER_IIC0_RXI
	PDL_INTC_REG_DTCER_IIC0_TXI

8) 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	Select the current CPU interrupt priority level or
PDL_INTC_REG_IR_(register) or	Interrupt Request register or
PDL_INTC_REG_IER_(register) or	Interrupt Request Enable register or
PDL_INTC_REG_IPR_(register) or	Interrupt Priority register or
PDL_INTC_REG_DTCER_(register)	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 69.

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
    );
}
```

9) 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	Select the current CPU interrupt priority level or
PDL_INTC_REG_IR_(register) or	Interrupt Request register or
PDL_INTC_REG_IER_(register) or	Interrupt Request Enable register or
PDL_INTC_REG_IPR_(register) or	Interrupt Priority register or
PDL_INTC_REG_DTCER_(register) or	DTC Activation Enable register or
PDL_INTC_REG_SWINTR	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 69.
- 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
    );
}
```

10) 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 69.

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 IER08 to 1 */
    R_INTC_Modify(
        PDL_INTC_REG_IER08,
        PDL_INTC_OR,
        0x50
    );
}
```

4.2.3. I/O Port

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

I/O port definitions

PDL_IO_PORT_0	Port P0	PDL_IO_PORT_5	Port P5	PDL_IO_PORT_D	Port PD
PDL_IO_PORT_1	Port P1	PDL_IO_PORT_A	Port PA	PDL_IO_PORT_E	Port PE
PDL_IO_PORT_2	Port P2	PDL_IO_PORT_B	Port PB	PDL_IO_PORT_H	Port PH
PDL_IO_PORT_3	Port P3	PDL_IO_PORT_C	Port PC	PDL_IO_PORT_J	Port PJ
PDL_IO_PORT_4	Port P4				

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_0_3	Port pin $P0_3$	PDL_IO_PORT_4_0	Port pin $P4_0$	PDL_IO_PORT_C_0	Port pin PC_0
PDL_IO_PORT_0_5	Port pin $P0_5$	PDL_IO_PORT_4_1	Port pin $P4_1$	PDL_IO_PORT_C_1	Port pin PC_1
PDL_IO_PORT_0_7	Port pin $P0_7$	PDL_IO_PORT_4_2	Port pin $P4_2$	PDL_IO_PORT_C_2	Port pin PC_2
		PDL_IO_PORT_4_3	Port pin $P4_3$	PDL_IO_PORT_C_3	Port pin PC_3
PDL_IO_PORT_1_2	Port pin $P1_2$	PDL_IO_PORT_4_4	Port pin $P4_4$	PDL_IO_PORT_C_4	Port pin PC_4
PDL_IO_PORT_1_3	Port pin $P1_3$	PDL_IO_PORT_4_5	Port pin $P4_5$	PDL_IO_PORT_C_5	Port pin PC_5
PDL_IO_PORT_1_4	Port pin $P1_4$	PDL_IO_PORT_4_6	Port pin $P4_6$	PDL_IO_PORT_C_6	Port pin PC_6
PDL_IO_PORT_1_5	Port pin $P1_5$	PDL_IO_PORT_4_7	Port pin $P4_7$	PDL_IO_PORT_C_7	Port pin PC_7
PDL_IO_PORT_1_6	Port pin $P1_6$				
PDL_IO_PORT_1_7	Port pin $P1_7$	PDL_IO_PORT_5_0	Port pin $P5_0$	PDL_IO_PORT_D_0	Port pin PD_0
		PDL_IO_PORT_5_1	Port pin $P5_1$	PDL_IO_PORT_D_1	Port pin PD_1
PDL_IO_PORT_2_0	Port pin $P2_0$	PDL_IO_PORT_5_2	Port pin $P5_2$	PDL_IO_PORT_D_2	Port pin PD_2
PDL_IO_PORT_2_1	Port pin $P2_1$	PDL_IO_PORT_5_3	Port pin $P5_3$	PDL_IO_PORT_D_3	Port pin PD_3
PDL_IO_PORT_2_2	Port pin $P2_2$	PDL_IO_PORT_5_4	Port pin $P5_4$	PDL_IO_PORT_D_4	Port pin PD_4
PDL_IO_PORT_2_3	Port pin $P2_3$	PDL_IO_PORT_5_5	Port pin $P5_5$	PDL_IO_PORT_D_5	Port pin PD_5
PDL_IO_PORT_2_4	Port pin $P2_4$			PDL_IO_PORT_D_6	Port pin PD_6
PDL_IO_PORT_2_5	Port pin $P2_5$	PDL_IO_PORT_A_0	Port pin PA_0	PDL_IO_PORT_D_7	Port pin PD_7
PDL_IO_PORT_2_6	Port pin $P2_6$	PDL_IO_PORT_A_1	Port pin PA_1		
PDL_IO_PORT_2_7	Port pin $P2_7$	PDL_IO_PORT_A_2	Port pin PA_2	PDL_IO_PORT_E_0	Port pin PE_0
		PDL_IO_PORT_A_3	Port pin PA_3	PDL_IO_PORT_E_1	Port pin PE_1
PDL_IO_PORT_3_0	Port pin $P3_0$	PDL_IO_PORT_A_4	Port pin PA_4	PDL_IO_PORT_E_2	Port pin PE_2
PDL_IO_PORT_3_1	Port pin $P3_1$	PDL_IO_PORT_A_5	Port pin PA_5	PDL_IO_PORT_E_3	Port pin PE_3
PDL_IO_PORT_3_2	Port pin $P3_2$	PDL_IO_PORT_A_6	Port pin PA_6	PDL_IO_PORT_E_4	Port pin PE_4
PDL_IO_PORT_3_3	Port pin $P3_3$	PDL_IO_PORT_A_7	Port pin PA_7	PDL_IO_PORT_E_5	Port pin PE_5
PDL_IO_PORT_3_4	Port pin $P3_4$			PDL_IO_PORT_E_6	Port pin PE_6
PDL_IO_PORT_3_5	Port pin $P3_5$	PDL_IO_PORT_B_0	Port pin PB_0	PDL_IO_PORT_E_7	Port pin PE_7
PDL_IO_PORT_3_6	Port pin $P3_6$	PDL_IO_PORT_B_1	Port pin PB_1		
PDL_IO_PORT_3_7	Port pin $P3_7$	PDL_IO_PORT_B_2	Port pin PB_2	PDL_IO_PORT_H_0	Port pin PH_0
		PDL_IO_PORT_B_3	Port pin PB_3	PDL_IO_PORT_H_1	Port pin PH_1
		PDL_IO_PORT_B_4	Port pin PB_4	PDL_IO_PORT_H_2	Port pin PH_2
		PDL_IO_PORT_B_5	Port pin PB_5	PDL_IO_PORT_H_3	Port pin PH_3
		PDL_IO_PORT_B_6	Port pin PB_6		
		PDL_IO_PORT_B_7	Port pin PB_7	PDL_IO_PORT_J_1	Port pin PJ_1
				PDL_IO_PORT_J_3	Port pin PJ_3

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
    uint16_t data2     // Configuration
);
```

Description

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. Use "|" to separate each selection.
Each selection is optional. If a selection is not made, the control setting will be left unchanged.

- Direction control

PDL_IO_PORT_INPUT or PDL_IO_PORT_OUTPUT	Input or output.

- Output type control

PDL_IO_PORT_TYPE_CMOS or PDL_IO_PORT_TYPE_NMOS or PDL_IO_PORT_TYPE_PMOS or PDL_IO_PORT_TYPE_HI_Z	Select CMOS push-pull output, N-channel open-drain, P-channel open-drain or high-impedance.	Available on selected pins. Available on pin PE1 only.

- Input pull-up resistor control

PDL_IO_PORT_PULL_UP_ON or PDL_IO_PORT_PULL_UP_OFF	On or off.

- Drive capacity control

PDL_IO_PORT_DRIVE_NORMAL or PDL_IO_PORT_DRIVE_HIGH	Normal or high-current drive. Valid for ports 1, B and C.

Return value

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

Category

I/O port

References

R_IO_PORT_NotAvailable

Remarks

- Ensure that the specified functions are valid for the selected port pin.
- The data direction and mode registers may be modified by other driver functions. Take care to not overwrite existing settings.
- Pin P35 is fixed as an input and cannot be modified.
- All pins that are not available on the selected package can be set to the required state using the R_IO_PORT_NotAvailable function.
- When a package with less than 100 pins has been chosen, the following port pins are mutually exclusive. If the user's code tries to configure both pins, the port pin selected in the second call of this function will be the one that is enabled.

48-pin package	64-pin package
PB0 and PC0	PB6 and PC0
PB1 and PC1	PB7 and PC1
PB3 and PC2	
PB5 and PC3	

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 pin P13 as an input port with the pull-up on */
    R_IO_PORT_Set(
        PDL_IO_PORT_1_3,
        PDL_IO_PORT_INPUT | PDL_IO_PORT_PULL_UP_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
    uint16_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.
PDL_IO_PORT_MODE or	General or Peripheral I/O mode control.
PDL_IO_PORT_TYPE or	Open-drain control.
PDL_IO_PORT_PULL_UP or	Pull-up control.
PDL_IO_PORT_DRIVE	Drive capacity control. Valid for ports 1, B and C.

[data3]
The address where the register value shall be stored, using one of the formats below.
Pin (not PE1 open-drain control):

b15 – b1	b0
0	0 or 1

Pin PE1 open-drain control:

b15 – b2	b1 – b0
0	0 to 3

Port (not open-drain control):

b15 – b8	b7 – b0
0	Register

Port (open-drain control)

b15 – b8	b7 – b0
Register ODR1	Register ODR0

Return value

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

Category

I/O port

References

None.

Remarks

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

Program example

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

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

void func(void)
{
    uint16_t result;

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

    /* Read the output type for pin P13 */
    R_IO_PORT_ReadControl(
        PDL_IO_PORT_1_3,
        PDL_IO_PORT_TYPE,
        &result
    );
}
```

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
    uint16_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.
PDL_IO_PORT_MODE or	General or Peripheral I/O mode control.
PDL_IO_PORT_TYPE or	Open-drain control.
PDL_IO_PORT_PULL_UP or	Pull-up control.
PDL_IO_PORT_DRIVE	Drive capacity control. Valid for ports 1, B and C.

- 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, using one of the formats below.
Pin (not PE1 open-drain) control:

b15 – b1	b0
Do not care	0 or 1

Pin PE1 open-drain control:

b15 – b2	b1 – b0
Do not care	0 to 3

Port (not open-drain) control:

b15 – b8	b7 – b0
Do not care	Register

Port (open-drain) control:

b15 – b8	b7 – b0
Register ODR1	Register ODR0

Return value

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

Category

I/O port

References

None.

Remarks

- Ensure that the specified functions are valid for the selected port pin.
- The data direction and mode registers may be modified by other driver Create functions. Take care to not overwrite existing settings.
- Pin P35 is fixed as an input and cannot be modified.

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_PULL_UP | 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

Remarks

- If an invalid port or pin is specified, the operation of the function cannot be guaranteed.
- The input buffer for the specified port or pin must be switched on (see R_IO_PORT_Set).

Program example

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

/* RPDL 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

None.

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
- For a port : 0x00 ~ 0xFF
- For a pin : 0 or 1

[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.
- The input buffer for the specified port or pin must be switched on (see R_IO_PORT_Set).

Program example

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

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

void IoHandler1{}
void IoHandler2{}

void func( void )
{
    /* Call function IoHandler1 if port pin P05 is high */
    R_IO_PORT_Compare(
        PDL_IO_PORT_0_5,
        1,
        IoHandler1
    );

    /* Call function IoHandler2 if port 5 reads as 0x55 */
    R_IO_PORT_Compare(
        PDL_IO_PORT_5,
        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
    uint16_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.
- For a port : 0x00 ~ 0xFF
- For a pin : 0 or 1

Return value

True if the parameters are valid; otherwise false.

Category

I/O port

References

None.

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 P05 */
    R_IO_PORT_Modify(
        PDL_IO_PORT_0_5,
        PDL_IO_PORT_XOR,
        1
    );

    /* And the value port 5 with 0x55 */
    R_IO_PORT_Modify(
        PDL_IO_PORT_5,
        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

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.
- The input buffer for the specified port or pin must be switched on (see R_IO_PORT_Set).

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 P05 reads as 0 */
    R_IO_PORT_Wait(
        PDL_IO_PORT_0_5,
        0
    );

    /* Wait until port 5 reads as 0x55 */
    R_IO_PORT_Wait(
        PDL_IO_PORT_5,
        0x55
    );
}
```

9) R_IO_PORT_NotAvailable

Synopsis

Configure I/O port pins that are not available.

Prototype

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

Description

Set the port pins that are not available on smaller packages to the recommended state.

Return value

True.

Category

I/O port

References

None.

Remarks

- All pins that are not available on the selected package will be configured for CMOS-type low-level output.

Program example

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

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

void func( void )
{
    /* Set all reserved I/O port pins to the recommended state */
    R_IO_PORT_NotAvailable();
}
```

4.2.4. Multifunction Pin Controller

The peripheral functions can be assigned to different pins, controlled by the Multifunction Pin Controller. The definitions available to the MPC functions are listed below.

MPC register definitions

PDL_MPC_REG_P07PFS	PDL_MPC_REG_PB0PFS
PDL_MPC_REG_P12PFS	PDL_MPC_REG_PB1PFS
PDL_MPC_REG_P13PFS	PDL_MPC_REG_PB2PFS
PDL_MPC_REG_P14PFS	PDL_MPC_REG_PB3PFS
PDL_MPC_REG_P15PFS	PDL_MPC_REG_PB4PFS
PDL_MPC_REG_P16PFS	PDL_MPC_REG_PB5PFS
PDL_MPC_REG_P17PFS	PDL_MPC_REG_PB6PFS
PDL_MPC_REG_P20PFS	PDL_MPC_REG_PB7PFS
PDL_MPC_REG_P21PFS	PDL_MPC_REG_PC0PFS
PDL_MPC_REG_P22PFS	PDL_MPC_REG_PC1PFS
PDL_MPC_REG_P23PFS	PDL_MPC_REG_PC2PFS
PDL_MPC_REG_P24PFS	PDL_MPC_REG_PC3PFS
PDL_MPC_REG_P25PFS	PDL_MPC_REG_PC4PFS
PDL_MPC_REG_P26PFS	PDL_MPC_REG_PC5PFS
PDL_MPC_REG_P27PFS	PDL_MPC_REG_PC6PFS
PDL_MPC_REG_P30PFS	PDL_MPC_REG_PC7PFS
PDL_MPC_REG_P31PFS	PDL_MPC_REG_PD0PFS
PDL_MPC_REG_P32PFS	PDL_MPC_REG_PD1PFS
PDL_MPC_REG_P33PFS	PDL_MPC_REG_PD2PFS
PDL_MPC_REG_P34PFS	PDL_MPC_REG_PD3PFS
PDL_MPC_REG_P40PFS	PDL_MPC_REG_PD4PFS
PDL_MPC_REG_P41PFS	PDL_MPC_REG_PD5PFS
PDL_MPC_REG_P42PFS	PDL_MPC_REG_PD6PFS
PDL_MPC_REG_P43PFS	PDL_MPC_REG_PD7PFS
PDL_MPC_REG_P44PFS	PDL_MPC_REG_PE0PFS
PDL_MPC_REG_P45PFS	PDL_MPC_REG_PE1PFS
PDL_MPC_REG_P46PFS	PDL_MPC_REG_PE2PFS
PDL_MPC_REG_P47PFS	PDL_MPC_REG_PE3PFS
PDL_MPC_REG_P54PFS	PDL_MPC_REG_PE4PFS
PDL_MPC_REG_P55PFS	PDL_MPC_REG_PE5PFS
PDL_MPC_REG_PA0PFS	PDL_MPC_REG_PE6PFS
PDL_MPC_REG_PA1PFS	PDL_MPC_REG_PE7PFS
PDL_MPC_REG_PA2PFS	PDL_MPC_REG_PH0PFS
PDL_MPC_REG_PA3PFS	PDL_MPC_REG_PH1PFS
PDL_MPC_REG_PA4PFS	PDL_MPC_REG_PH2PFS
PDL_MPC_REG_PA5PFS	PDL_MPC_REG_PH3PFS
PDL_MPC_REG_PA6PFS	PDL_MPC_REG_PJ1PFS
PDL_MPC_REG_PA7PFS	PDL_MPC_REG_PJ3PFS

1) R_MPC_Read

Synopsis

Read an MPC register.

Prototype

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

Description

Get the value of an MPC register.

[data1]
One of the definition values from §4.2.4.

[data2]
The value read from the register.

Return value

True if a valid MPC register is specified; otherwise false.

Category

MPC registers

References

None.

Remarks

- None.

Program example

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

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

void func( void )
{
    uint8_t data;

    /* Get the value of register P07PFS */
    R_MPC_Read(
        PDL_MPC_REG_P07PFS,
        &data
    );
}
```

2) R_MPC_Write

Synopsis

Write to a MPC register.

Prototype

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

Description

Write the value to an MPC 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 MPC register is specified; otherwise false.

Category

MPC registers

References

None.

Remarks

- The MPC registers are modified by other driver functions. Take care to not overwrite existing settings.
- Refer to the hardware manual for valid values for each register.

Program example

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

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

void func( void )
{
     /* Write data to register PD1PFS */
    R_MPC_Write(
        PDL_MPC_REG_PD1PFS,
        0xFF
    );
}
```

3) R_MPC_Modify

Synopsis

Modify an MPC register.

Prototype

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

Description

Write the value to an MPC register.

[data1]
One of the definition values from §4.2.4.

[data2]
- The logical operation to be applied to the register contents.

PDL_MPC_AND or PDL_MPC_OR or PDL_MPC_XOR	Select between AND (&), OR (|) or Exclusive-OR (^).

[data3]
The value to be used for the modification.

Return value

True if a valid MPC register is specified; otherwise false.

Category

MPC registers

References

None.

Remarks

- The MPC registers are modified by other driver functions. Take care to not overwrite existing settings.
- Refer to the hardware manual for valid values for each register.

Program example

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

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

void func( void )
{
    /* Set bit 6 in P12PFS to 1 */
    R_MPC_Modify(
        PDL_MPC_REG_P12PFS,
        PDL_MPC_OR,
        0x40
    );
}
```

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

- Software reset control

PDL_MCU_RESET_START	Start a software reset of the MCU.

- Start type flag control

PDL_MCU_WARM_START	Set the Start type status flag to Warm.

Return value

True if a valid register is specified; otherwise false.

Category

MCU registers

References

R_CGC_Set, R_RTC_Create

Remarks

- The PDL_MCU_WARM_START is used after the initialization of cold start (caused by a power-on reset) has completed. This is to indicate the next reset processing is warm start. (Caused by a reset signal during operation).

Program example

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

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

void func( void )
{
    /* Reset the MCU */
    R_MCU_Control(
        PDL_MCU_RESET_START
    );
}
```

2) R_MCU_GetStatus

Synopsis

Read the MCU status.

Prototype

```
bool R_MCU_GetStatus(
    uint16_t * data1,   // The location where the mode status flags shall be stored
    uint16_t * data2,   // The location where the reset status flags shall be stored
    uint32_t * data3,   // The storage location for the Option Function Select Register 0
    uint32_t * data4    // The storage location for the Option Function Select Register 1
);
```

Description

Read the status registers for the MCU.

[data1]
The status flags shall be stored in the format below.
Specify PDL_NO_PTR if they are not required.

b15 – b14	b13	b12 – b9	b8
0	User boot mode 0: Other 1: Selected	0	1

b7 – b5	b4 – b1	b0
Endian mode 000b: Big 111b: Little	0	MD pin level at release from reset 0: Low 1: High

[data2]
The reset status flags shall be stored in the format below.
Specify PDL_NO_PTR if they are not required.

b15 – b9	b8
0	Start type 0: Cold 1: Warm

b7	b6	b5	b4	b3	b2	b1	b0
Reset detection flags							
0	Software	0	IWDT	Voltage monitor			Power-on
				2	1	0	
0: Not detected 1: detected							

[data3]
Where the OFS0 register contents shall be stored.
Please refer to the MCU hardware manual for the format.
Specify PDL_NO_PTR if they are not required.

[data4]
Where the OFS1 register contents shall be stored.
Please refer to the MCU hardware manual for the format.
Specify PDL_NO_PTR if they are not required.

Return value

True.

Category

MCU registers

References

None.

Remarks

- If a reset detection flag is set to 1, it shall be automatically cleared to 0 by this function.

Program example

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

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

void func( void )
{
    uint16_t status;

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

3) R_MCU_OFS

Synopsis

Configure the device start-up operation.

Prototype

```
R_MCU_OFS(
    uint32_t data1,    // IWDT configuration options
    uint32_t data2,    // LVD configuration options
    uint32_t data3     // CGC configuration options
);
```

Description (1/2)

Select the auto-start settings to be stored in registers OFS0 and OFS1.

[data1]
Select the post-reset IWDT configuration settings.
If multiple selections are required, use "|" to separate each selection.

- Auto-start control

PDL_MCU_OFS_IWDT_HALTED or PDL_MCU_OFS_IWDT_AUTOSTART	Disable or enable the IWDT auto-start mode.

If auto-start mode is enabled, select one setting from each of the following.

- Timeout period

PDL_MCU_OFS_IWDT_TIMEOUT_1024 or PDL_MCU_OFS_IWDT_TIMEOUT_4096 or PDL_MCU_OFS_IWDT_TIMEOUT_8192 or PDL_MCU_OFS_IWDT_TIMEOUT_16384	Timeout period specified in cycles of the divided clock as specified in the Clock division selection below.

- Clock division

PDL_MCU_OFS_IWDT_CLOCK_LOCO_1 or PDL_MCU_OFS_IWDT_CLOCK_LOCO_16 or PDL_MCU_OFS_IWDT_CLOCK_LOCO_32 or PDL_MCU_OFS_IWDT_CLOCK_LOCO_64 or PDL_MCU_OFS_IWDT_CLOCK_LOCO_128 or PDL_MCU_OFS_IWDT_CLOCK_LOCO_256	The selected clock. The LOCO ÷ 1, 16, 32, 64, 128 or 256.

- Window end position

PDL_MCU_OFS_IWDT_WIN_END_75 or PDL_MCU_OFS_IWDT_WIN_END_50 or PDL_MCU_OFS_IWDT_WIN_END_25 or PDL_MCU_OFS_IWDT_WIN_END_0	The window end position specified as a percentage of the down-counter. 0% is when the down-counter would underflow. Selecting 0% is equivalent to no window end position.

- Window start position

PDL_MCU_OFS_IWDT_WIN_START_25 or PDL_MCU_OFS_IWDT_WIN_START_50 or PDL_MCU_OFS_IWDT_WIN_START_75 or PDL_MCU_OFS_IWDT_WIN_START_100	The window start position specified as a percentage of the down-counter. 0% is when the down-counter would underflow. Selecting 100% is equivalent to no window start position.

- Underflow action

PDL_MCU_OFS_IWDT_NMI or PDL_MCU_OFS_IWDT_RESET	Select an NMI or reset when the IWDT down-counter underflows.

- Count stop mode

PDL_MCU_OFS_IWDT_STOP_DISABLE or PDL_MCU_OFS_IWDT_STOP_ENABLE	Enable or disable Count stop mode. If the Count Stop mode is enabled the IWDT counter is stopped at a transition to sleep mode, software standby mode, deep software standby mode, or all-module clock stop mode.

Description (2/2)

[data2]
Select the post-reset LVD configuration settings.

- Auto-start control

PDL_MCU_OFS_LVD_0_DISABLE or PDL_MCU_OFS_LVD_0_ENABLE	Disable or enable the Voltage monitor 0 auto-start mode.

If auto-start mode is enabled, select one setting from the following.

- Voltage threshold selection

PDL_MCU_OFS_LVD_0_380 or PDL_MCU_OFS_LVD_0_280 or PDL_MCU_OFS_LVD_0_190 or PDL_MCU_OFS_LVD_0_172	Select 3.80V, 2.80V, 1.90V or 1.72V as the voltage monitor 0 detection level.

[data3]
Select the post-reset CGC configuration settings.

- Auto-start control

PDL_MCU_OFS_CGC_HOCO_DISABLE or PDL_MCU_OFS_CGC_HOCO_ENABLE	Disable or enable the HOCO after a reset.

Category

MCU registers

References

R_IWDT_Set, R_CGC_Set

Remarks

- This is a macro, not a function call. There is no error checking or return value.
- The auto-start setting for each parameter must be selected.

Program example

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

/* Enable the IWDT auto-start mode. */
/* Enable reset at Vdet0. */
/* Leave the HOCO disabled. */
R_MCU_OFS(
    PDL_MCU_OFS_IWDT_AUTOSTART | PDL_MCU_OFS_IWDT_TIMEOUT_4096 | \
    PDL_MCU_OFS_IWDT_CLOCK_LOCO_16|\
    PDL_MCU_OFS_IWDT_WIN_END_50 |\
    PDL_MCU_OFS_IWDT_WIN_START_75 | PDL_MCU_OFS_IWDT_NMI | \
    PDL_MCU_OFS_IWDT_STOP_DISABLE,
    PDL_MCU_OFS_LVD_0_ENABLE,
    PDL_MCU_OFS_CGC_HOCO_DISABLE
);
```

4.2.6. Voltage Detection Circuit

1) R_LVD_Create

Synopsis

Configure the voltage detection circuit.

Prototype

```
bool R_LVD_Create(
    uint16_t data1,     // Monitor 1 Configuration selection
    uint16_t data2,     // Monitor 1 Voltage selection
    uint16_t data3,     // Monitor 2 Configuration selection
    uint16_t data4,     // Monitor 2 Voltage selection
    void * func1,       // Monitor 1 Callback function
    uint8_t data5,      // Monitor 1 Interrupt priority level
    void * func2,       // Monitor 2 Callback function
    uint8_t data6       // Monitor 2 Interrupt priority level
);
```

Description (1/2)

Set the voltage detection configuration.

[data1]
Monitor 1 voltage detection configuration.
If the monitor is not required specify PDL_NO_DATA, otherwise use “|” to separate each selection.

- Operation.

PDL_LVD_MONITOR_ONLY or PDL_LVD_RESET_NEGATION_VCC_MORE_THAN_VDET or PDL_LVD_RESET_NEGATION_AFTER_DELAY or PDL_LVD_INTERRUPT_NMI_DETECT_RISE or PDL_LVD_INTERRUPT_NMI_DETECT_FALL or PDL_LVD_INTERRUPT_NMI_DETECT_RISE_AND_FALL or PDL_LVD_INTERRUPT_MI_DETECT_RISE or PDL_LVD_INTERRUPT_MI_DETECT_FALL or PDL_LVD_INTERRUPT_MI_DETECT_RISE_AND_FALL	Select no action, a reset on low voltage detection or an interrupt when a specified voltage event is detected.

- Digital Filter

PDL_LVD_FILTER_DISABLE or PDL_LVD_FILTER_LOCO_DIV_1 or PDL_LVD_FILTER_LOCO_DIV_2 or PDL_LVD_FILTER_LOCO_DIV_4 or PDL_LVD_FILTER_LOCO_DIV_8	Configure the digital filter.

[data2]
Monitor 1 voltage detection level. Specify PDL_NO_DATA if not required.

PDL_LVD_VOLTAGE_LEVEL_415 or PDL_LVD_VOLTAGE_LEVEL_400 or PDL_LVD_VOLTAGE_LEVEL_385 or PDL_LVD_VOLTAGE_LEVEL_370 or PDL_LVD_VOLTAGE_LEVEL_355 or PDL_LVD_VOLTAGE_LEVEL_340 or PDL_LVD_VOLTAGE_LEVEL_325 or PDL_LVD_VOLTAGE_LEVEL_310 or PDL_LVD_VOLTAGE_LEVEL_295 or PDL_LVD_VOLTAGE_LEVEL_280 or PDL_LVD_VOLTAGE_LEVEL_265 or PDL_LVD_VOLTAGE_LEVEL_250 or PDL_LVD_VOLTAGE_LEVEL_235 or PDL_LVD_VOLTAGE_LEVEL_220 or PDL_LVD_VOLTAGE_LEVEL_205 or PDL_LVD_VOLTAGE_LEVEL_190	Set the voltage detection level. For example PDL_LVD_VOLTAGE_LEVEL_415 = 4.15V. Required only if the monitor is enabled.

Description (2/2)

[data3]
Monitor 2 voltage detection configuration.
If the monitor is not required specify PDL_NO_DATA, otherwise use “|” to separate each selection.

- Operation

PDL_LVD_MONITOR_ONLY or PDL_LVD_RESET_NEGATION_VCC_MORE_THAN_VDET or PDL_LVD_RESET_NEGATION_AFTER_DELAY or PDL_LVD_INTERRUPT_NMI_DETECT_RISE or PDL_LVD_INTERRUPT_NMI_DETECT_FALL or PDL_LVD_INTERRUPT_NMI_DETECT_RISE_AND_FALL or PDL_LVD_INTERRUPT_MI_DETECT_RISE or PDL_LVD_INTERRUPT_MI_DETECT_FALL or PDL_LVD_INTERRUPT_MI_DETECT_RISE_AND_FALL	Select no action, a reset on low voltage detection or an interrupt when a specified voltage event is detected.

- Digital Filter

PDL_LVD_FILTER_DISABLE or PDL_LVD_FILTER_LOCO_DIV_1 or PDL_LVD_FILTER_LOCO_DIV_2 or PDL_LVD_FILTER_LOCO_DIV_4 or PDL_LVD_FILTER_LOCO_DIV_8	Configure the digital filter.

- Pin selection

PDL_LVD_VDET2_PIN_VCC or PDL_LVD_VDET2_PIN_CMPA2	Monitor VCC or the CMPA2 pin.

[data4]
Monitor 2 voltage detection level. Specify PDL_NO_DATA if not required.

PDL_LVD_VOLTAGE_LEVEL_415 or PDL_LVD_VOLTAGE_LEVEL_400 or PDL_LVD_VOLTAGE_LEVEL_385 or PDL_LVD_VOLTAGE_LEVEL_370 or PDL_LVD_VOLTAGE_LEVEL_355 or PDL_LVD_VOLTAGE_LEVEL_340 or PDL_LVD_VOLTAGE_LEVEL_325 or PDL_LVD_VOLTAGE_LEVEL_310 or PDL_LVD_VOLTAGE_LEVEL_295 or PDL_LVD_VOLTAGE_LEVEL_280 or PDL_LVD_VOLTAGE_LEVEL_265 or PDL_LVD_VOLTAGE_LEVEL_250 or PDL_LVD_VOLTAGE_LEVEL_235 or PDL_LVD_VOLTAGE_LEVEL_220 or PDL_LVD_VOLTAGE_LEVEL_205 or PDL_LVD_VOLTAGE_LEVEL_190	Set the voltage detection level. For example PDL_LVD_VOLTAGE_LEVEL_415 = 4.15V. Required only if the monitor is enabled and the VCC pin is selected.

[func1]
The function to be called when a Monitor 1 maskable interrupt occurs.
Specify PDL_NO_FUNC if not required.

[data5]
The interrupt priority level for the Monitor 1 interrupt.
If specifying a callback function in func1 then select between 1 (lowest priority) and 15 (highest priority).Set to 0 if using LVD to trigger the ELC without generating an interrupt, see Remarks for details.

[func2]
The function to be called when a Monitor 2 maskable interrupt occurs.
Specify PDL_NO_FUNC if not required.

[data6]
The interrupt priority level for the Monitor 2 interrupt.
If specifying a callback function in func2 then select between 1 (lowest priority) and 15 (highest priority).

Return value True if the parameters are valid; otherwise false.

Category Voltage detection circuit.

References R_INTC_CreateExtInterrupt, R_CGC_Set, R_CGC_Control, R_LPC_GetStatus, R_MCU_OFS

Remarks

- If a non-maskable interrupt will be generated, call R_INTC_CreateExtInterrupt to set up the NMI handler and to accept LVD-based interrupt signals.
- The LVD shares its registers with Comparator A, so both cannot be used at the same time.
- If using the digital filter, function R_CGC_Set must be called (with the current clock source selected) before using this function.
- If using the digital filter the LOCO clock must be enabled. Use R_CGC_Set (with the LOCO selected).
- Following a reset, function R_LPC_GetStatus can be used to see what caused the reset.
- If using a delay on Reset negation then the LOCO clock must be enabled. See R_CGC_Set or R_CGC_Control.
- If the CMPA2 pin input is selected, the detection voltage is fixed at 1.33V.
- The same voltage level must not be specified for more than one voltage monitor. This includes voltage monitor 0 (see R_MCU_OFS). If this condition is detected this function will return false.
- To enable the LVD event link output function, enable the LVD first then enable the LVD event link function at the ELC. To disable this function, disable the LVD event link function at the ELC first then disable the LVD.
- It is possible to configure the LVD to trigger the ELC but not generate an interrupt itself. To do this setup the LVD as required using one of the following operations:
  PDL_LVD_INTERRUPT_MI_DETECT_RISE or
  PDL_LVD_INTERRUPT_MI_DETECT_FALL or
  PDL_LVD_INTERRUPT_MI_DETECT_RISE_AND_FALL
  Set the callback function as PDL_NO_FUNC and set the interrupt priority to 0.
- Disable the digital filter circuit when using voltage monitoring 1 and 2 circuit in software standby mode.
- ELC is only valid for voltage detection 1.
- User wants to use both LVD1 and LVD2: user must configure both LVD1 and LVD2 simultaneously.

Program example

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

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

void Callback_LowVoltage(void);

void func( void )
{
    /* Use Monitor 2 to generate an NMI when VCC drops below 4V.*/
    R_LVD_Create(
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_LVD_INTERRUPT_NMI_DETECT_FALL | \
        PDL_LVD_FILTER_DISABLE | PDL_LVD_VDET2_PIN_VCC,
        PDL_LVD_VOLTAGE_LEVEL_400, //4.00V
        PDL_NO_FUNC,
        PDL_NO_DATA,
        Callback_LowVoltage,
        15
    );
}
```

2) R_LVD_Control

Synopsis

Control the voltage detection circuit.

Prototype

```
bool R_LVD_Control(
    uint8_t data1,      // Monitor 1 control
    uint8_t data2       // Monitor 2 control
);
```

Description

Control the voltage detection configuration.

[data1]
Monitor 1 control. All selections are optional.
If multiple selections are required, use "|" to separate each selection.
If no selections are required, specify PDL_NO_DATA.

- Monitor control

PDL_LVD_DISABLE	Disable monitor 1 operation.

- Flag control

PDL_LVD_CLEAR_DETECTION	Clear the monitor 1 change detection flag.

[data2]
Monitor 2 control. All selections are optional.
If multiple selections are required, use "|" to separate each selection.
If no selections are required, specify PDL_NO_DATA.

- Monitor control

PDL_LVD_DISABLE	Disable monitor 2 operation.

- Flag control

PDL_LVD_CLEAR_DETECTION	Clear the monitor 2 change detection flag.

Return value

True.

Category

Voltage detection circuit

References

R_LVD_Create

Remarks

- Other operation changes require the shutdown of both voltage monitors. If such changes are required, call R_LVD_Create with the new settings.

Program example

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

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

void func( void )
{
    /* Disable monitor 1; clear the monitor 2 flag  */
    R_LVD_Control(
        PDL_LVD_DISABLE,
        PDL_LVD_CLEAR_DETECTION
    );
}
```

3) R_LVD_GetStatus

Synopsis

Check the status of the voltage detection module.

Prototype

```
bool R_LVD_GetStatus(
        uint8_t * data      // Status flags pointer
);
```

Description

Return the status flags.

[data]
The Monitor 1 and Monitor 2 status flag shall be stored in the following format.

b7 - b6	b5	b4	b3 - b2	b1	b0
	Monitor 2			Monitor 1	
	Status	Change		Status	Change
0	0: VCC < Vdet2 1: VCC ≥ Vdet2, or the monitor is disabled	0: None 1: Detected	0	0: VCC < Vdet1 1: VCC ≥ Vdet1, or the monitor is disabled.	0: None 1: Detected

Return value

True.

Category

LVD

Reference

R_LVD_Control, R_LVD_Create

Remarks

- Use R_LVD_Control to clear the detection flags.
- A detection flag is not valid if Monitor-only operation was selected in R_LVD_Create.

Program example

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

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

void func(void)
{
    uint8_t StatusFlags;

    /* Read the LVD status */
    R_LVD_GetStatus(
        &StatusFlags
    );
}
```

4.2.7. Clock Frequency Accuracy Measurement Circuit

1) R_CAC_Create

Synopsis

Configure the clock accuracy circuit.

Prototype

```
bool R_CAC_Create(
    uint32_t data1,    // Signal selection
    uint8_t data2,     // External input configuration
    double data3,      // External input timing
    uint16_t data4,    // Upper-limit value
    uint16_t data5,    // Lower-limit value
    void* func1,       // Callback function
    uint8_t data6,     // Interrupt priority level
    void* func2,       // Callback function
    uint8_t data7,     // Interrupt priority level
    void* func3,       // Callback function
    uint8_t data8      // Interrupt priority level
);
```

Description (1/2)

Configure the operation of the Clock frequency accuracy measurement circuit.

[data1]
Choose the reference and measure clock settings. Use "|" to separate each selection.

- Reference signal selection

PDL_CAC_REFERENCE_MAIN or PDL_CAC_REFERENCE_SUB_CLOCK or PDL_CAC_REFERENCE_HOCO or PDL_CAC_REFERENCE_LOCO or PDL_CAC_REFERENCE_IWDTLOCO or PDL_CAC_REFERENCE_CACREF	Select the main clock oscillator, sub-clock oscillator, high-speed on-chip oscillator, low-speed on-chip oscillator or IWDT low-speed on-chip oscillator or input to pin CACREF as the reference signal.
PDL_CAC_REFERENCE_RISING or PDL_CAC_REFERENCE_FALLING or PDL_CAC_REFERENCE_BOTH	Select rising edges, falling edges or both rising and falling edges to be valid.
PDL_CAC_REFERENCE_DIV_32 or PDL_CAC_REFERENCE_DIV_128 or PDL_CAC_REFERENCE_DIV_1024 or PDL_CAC_REFERENCE_DIV_8192	Divide the reference signal by 32, 128, 1024 or 8192. Not required when the CACREF input is selected as the reference signal.

- Measured clock selection and division

PDL_CAC_MEASURE_MAIN or PDL_CAC_MEASURE_SUB_CLOCK or PDL_CAC_MEASURE_HOCO or PDL_CAC_MEASURE_LOCO or PDL_CAC_MEASURE_IWDTLOCO	Select the main clock oscillator, sub-clock oscillator, high-speed on-chip oscillator, low-speed on-chip oscillator or IWDT low-speed on-chip oscillator for measurement.
PDL_CAC_MEASURE_DIV_1 or PDL_CAC_MEASURE_DIV_4 or PDL_CAC_MEASURE_DIV_8 or PDL_CAC_MEASURE_DIV_32	Divide the clock to be measured by 1, 4, 8 or 32.

- Limit value calculation

PDL_CAC_LIMIT_TOLERANCE or PDL_CAC_LIMIT_REGISTER	Parameters data4 and data5 will contain either the tolerance or the limit register values.

Description (2/2)

[data2]
Choose the CACREF input settings. Use "|" to separate each selection.
If the CACREF input is not required, specify PDL_NO_DATA.

- External input configuration

PDL_CAC_CACREF_PORT_A_0 or PDL_CAC_CACREF_PORT_C_7 or PDL_CAC_CACREF_PORT_H_0	Select the pin to be used for signal CACREF. Parameter data3 contains the frequency of the signal applied to this pin.
PDL_CAC_CACREF_FILTER_DISABLE or PDL_CAC_CACREF_FILTER_DIV_1 or PDL_CAC_CACREF_FILTER_DIV_4 or PDL_CAC_CACREF_FILTER_DIV_16	If used, the CACREF signal can be unfiltered or sampled using the clock to be measured divided by 1, 4 or 16.

[data3]
If the CACREF input will be used, specify the input clock frequency (in Hz).
Use PDL_NO_DATA if not required.

[data4]
Specify either:
a) the maximum positive deviation for the measured clock as a percentage, or
b) the upper count limit for the measured clock where the maximum value is 65535.

[data5]
Specify either:
a) the maximum negative deviation for the measured clock as a percentage, or
b) the lower count limit for the measured clock where the maximum value is 65535.

[func1]
The function to be called if a frequency error is detected.
Specify PDL_NO_FUNC if not required.

[data6]
The frequency error 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 func1.

[func2]
The function to be called when the measurement has ended.
Specify PDL_NO_FUNC if not required.

[data7]
The measurement complete 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 func2.

[func3]
The function to be called if the measurement counter overflows.
Specify PDL_NO_FUNC if not required.

[data8]
The counter overflow 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 func3.

Return value

True if all parameters are valid and exclusive and the selected clocks have been set; otherwise false.

Category

Clock frequency accuracy measurement circuit

References

R_CGC_Set

Remarks

- If the external input CACREF pin is selected, the Multifunction Pin Control registers are modified to enable the selected pin.
- Before using this function, call R_CGC_Set for each clock that will be measured or used as a reference.
- If both edges are selected, the clock duty cycle is assumed to be 50%.
- If a frequency error callback function is specified then it must clear the error flag, using R_CAC_Control, to prevent continuous interrupts / callbacks.
- If a measurement complete callback function is specified then it must clear the measurement flag, using R_CAC_Control, to prevent continuous interrupts / callbacks.
- If an overflow callback function is specified then it must clear the overflow flag, using R_CAC_Control, to prevent continuous interrupts / callbacks.

Program example

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

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

/* Callback functions */
void CAC_frequency_error( void ){}
void CAC_measurement_complete( void ){}
void CAC_overflow( void ){}

void func(void)
{
    /* Use the main clock to check the LOCO accuracy */
    R_CAC_Create(
        PDL_CAC_REFERENCE_MAIN | PDL_CAC_REFERENCE_RISING | \
        PDL_CAC_REFERENCE_DIV_8192 | \
        PDL_CAC_MEASURE_LOCO | PDL_CAC_MEASURE_DIV_1 | \
        PDL_CAC_LIMIT_TOLERANCE,
        PDL_NO_DATA,
        PDL_NO_DATA,
        10,
        10,
        CAC_frequency_error,
        15,
        CAC_measurement_complete,
        6,
        CAC_overflow,
        10
    );
}
```

2) R_CAC_Destroy

Synopsis	Stop the clock accuracy circuit.
Prototype	**bool R_CAC_Destroy(** **void** // No parameter is required **);**
Description	Disable and shutdown the Clock frequency accuracy measurement circuit.
Return value	True.
Category	Clock frequency accuracy measurement circuit
Reference	None.
Remarks	• The CAC module is halted, to reduce the current consumption.
Program example	

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

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

void func(void)
{
    /* Disable the CAC */
    R_CAC_Destroy(
    );
}
```

3) R_CAC_Control

Synopsis

Control the clock accuracy circuit.

Prototype

```
bool R_CAC_Control(
    uint8_t data1,     // Control options
    uint32_t data2,    // Operation changes
    uint16_t data3,    // Upper-limit value
    uint16_t data4     // Lower-limit value
);
```

Description (1/2)

Modify the Clock frequency accuracy measurement circuit operation.

[data1]
Control options. All selections are optional.
If multiple selections are required, use "|" to separate each selection.
If no selections are required, specify PDL_NO_DATA.

- Flag clearing control

PDL_CAC_CLEAR_FREQUENCY_ERROR	Clear any selected flag.
PDL_CAC_CLEAR_MEASUREMENT	
PDL_CAC_CLEAR_OVERFLOW	

- Operation control

PDL_CAC_DISABLE	Stop the measurement operation.
PDL_CAC_ENABLE	Re-start the measurement operation.

[data2]
Operation control options. All selections are optional.
If multiple selections are required, use "|" to separate each selection.
If no selections are required, specify PDL_NO_DATA.

- Reference signal selection

PDL_CAC_REFERENCE_MAIN or PDL_CAC_REFERENCE_SUB_CLOCK or PDL_CAC_REFERENCE_HOCO or PDL_CAC_REFERENCE_LOCO or PDL_CAC_REFERENCE_IWDTLOCO or PDL_CAC_REFERENCE_CACREF	Select the main clock oscillator, sub-clock oscillator, high-speed on-chip oscillator, low-speed on-chip oscillator or IWDT low-speed on-chip oscillator or input to pin CACREF as the reference signal.

- Reference signal edge selection

PDL_CAC_REFERENCE_RISING or PDL_CAC_REFERENCE_FALLING or PDL_CAC_REFERENCE_BOTH	Select rising edges, falling edges or both rising and falling edges to be valid.

- Reference signal division selection

PDL_CAC_REFERENCE_DIV_32 or PDL_CAC_REFERENCE_DIV_128 or PDL_CAC_REFERENCE_DIV_1024 or PDL_CAC_REFERENCE_DIV_8192	If an internal clock is used as the reference signal, divide it by 32, 128, 1024 or 8192. Ignored if the CACREF input is selected.

- Measured clock selection

PDL_CAC_MEASURE_MAIN or PDL_CAC_MEASURE_SUB_CLOCK or PDL_CAC_MEASURE_HOCO or PDL_CAC_MEASURE_LOCO or PDL_CAC_MEASURE_IWDTLOCO	Select the main clock oscillator, sub-clock oscillator, high-speed on-chip oscillator, low-speed on-chip oscillator or IWDT low-speed on-chip oscillator for measurement.

Description (2/2)

- Measured clock division selection

PDL_CAC_MEASURE_DIV_1 or PDL_CAC_MEASURE_DIV_4 or PDL_CAC_MEASURE_DIV_8 or PDL_CAC_MEASURE_DIV_32	Divide the clock to be measured by 1, 4, 8 or 32.

- Limit value calculation

PDL_CAC_LIMIT_TOLERANCE or PDL_CAC_LIMIT_REGISTER	Parameters data3 and data4 will contain either the tolerance or the limit register values.

[data3]
If selected in parameter data2, specify either:
a) the maximum positive deviation for the measured clock as a percentage, or
b) the upper count limit for the measured clock where the maximum value is 65535.
If not required, specify PDL_NO_DATA.

[data4]
If selected in parameter data2, specify either:
a) the maximum negative deviation for the measured clock as a percentage, or
b) the lower count limit for the measured clock where the maximum value is 65535.
If not required, specify PDL_NO_DATA.

Return value

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

Category

Clock frequency accuracy measurement circuit

References

R_CAC_Create

Remarks

- If signal selection or limit value changes are required, the measurement operation must be disabled.
- The Disable operation is executed at the start of this function.
  The Enable operation is executed at the end.
  Therefore, both options can be selected together with operation changes in one function call.
- If the Disable and/or Enable operation is selected, this function will wait for the operation to complete before continuing. To prevent lockup, ensure that an enable / disable operation is not also performed from a callback function at the same time.
- If the CACREF input is selected, the digital filter setting used in R_CAC_Create will be retained.

Program example

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

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

void func(void)
{
    /* Clear the measurement-complete flag (without stopping) */
    R_CAC_Control(
        PDL_CAC_CLEAR_MEASUREMENT,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA
    );
}
```

4) R_CAC_GetStatus

Synopsis

Read the clock accuracy circuit status.

Prototype

```
bool R_CAC_GetStatus(
    uint8_t * data1,    // Pointer to the variable where the status value shall be stored.
    uint16_t * data2,   // Data storage location
    uint16_t * data3,   // Data storage location
    uint16_t * data4    // Data storage location
);
```

Description

Read the status, limit and counter registers.

[data1]
The status flags shall be stored in the format below.

b7 – b4	b3	b2	b1	b0
0	Overflow 0: Not detected 1: Detected.	Measurement 0: No event 1: Completed	Frequency error 0: Not detected 1: Detected	Operation 0: Disabled 1: Enabled

[data2]
Where the upper-limit value register (CAULVR) value shall be stored.
Specify PDL_NO_PTR if it is not required.

[data3]
Where the lower-limit value register (CALLVR) value shall be stored.
Specify PDL_NO_PTR if it is not required.

[data4]
Where the counter buffer register (CACNTBR) value shall be stored.
Specify PDL_NO_PTR if it is not required.

Return value

True.

Category

Clock frequency accuracy measurement circuit

References

None.

Remarks

- None.

Program example

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

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

void func(void)
{
    uint8_t Status_flags;

    R_CAC_GetStatus(
        &Status_flags,
        PDL_NO_PTR,
        PDL_NO_PTR,
        PDL_NO_PTR
    );
}
```

4.2.8. Low Power Consumption

1) R_LPC_Create

Synopsis

Configure the MCU low power conditions.

Prototype

```
bool R_LPC_Create(
    uint32_t data1,    // Configuration options
    uint16_t data2,    // Main oscillator waiting time
    uint16_t data3,    // Sub-clock oscillator waiting time
    uint32_t data4     // HOCO oscillator waiting time
);
```

Description (1/3)

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.

- Operating power control

PDL_LPC_MIDDLE_SPEED_MODE_A or PDL_LPC_MIDDLE_SPEED_MODE_B or PDL_LPC_LOW_SPEED_MODE_1 or PDL_LPC_LOW_SPEED_MODE_2	Select the operating power control mode.

- Sleep mode return clock source switching

PDL_LPC_SLEEP_RETURN_SWITCH_DISABLE or PDL_LPC_SLEEP_RETURN_SWITCH_HOCO or PDL_LPC_SLEEP_RETURN_SWITCH_MAIN	Control clock source switching at cancellation of sleep mode.

- Flash HOCO software standby control

PDL_LPC_SOFTCUT_POR or	Power is supplied to HOCO in software standby mode. The voltage detection circuit (LVD) is active and the low power consumption function by the power-on reset circuit (POR) is disabled.
PDL_LPC_SOFTCUT_HOCO_POR or	Power is not supplied to HOCO in software standby mode. The voltage detection circuit (LVD) is active and the low power consumption function by the power-on reset circuit (POR) is disabled.
PDL_LPC_SOFTCUT_LVD or	Power is supplied to HOCO in software standby mode. The voltage detection circuit (LVD) is stopped and the power consumption reduction function by the power-on reset circuit (POR) is enabled.
PDL_LPC_SOFTCUT_HOCO_LVD	Power is not supplied to HOCO in software standby mode. The voltage detection circuit (LVD) is stopped and the power consumption reduction function by the power-on reset circuit (POR) is enabled.

Description (2/3)

[data2]
Select the main clock oscillator waiting time.
If no selections are required, specify PDL_NO_DATA.

- Main clock oscillator waiting time

PDL_LPC_MAIN_2 or PDL_LPC_MAIN_4 or PDL_LPC_MAIN_8 or PDL_LPC_MAIN_16 or PDL_LPC_MAIN_32 or PDL_LPC_MAIN_256 or PDL_LPC_MAIN_512 or PDL_LPC_MAIN_1024 or PDL_LPC_MAIN_2048 or PDL_LPC_MAIN_4096 or PDL_LPC_MAIN_16384 or PDL_LPC_MAIN_32768 or PDL_LPC_MAIN_65536 or PDL_LPC_MAIN_131072 or PDL_LPC_MAIN_262144 or PDL_LPC_MAIN_524288	Select the oscillation settling time of the main clock oscillator before the CPU resumes after exiting from software standby mode. When updating this value, the main clock oscillator must be stopped.

[data3]
Select the sub clock oscillator waiting times.
If no selections are required, specify PDL_NO_DATA.

- Sub clock oscillator waiting time

PDL_LPC_SUB_2 or PDL_LPC_SUB_4 or PDL_LPC_SUB_8 or PDL_LPC_SUB_16 or PDL_LPC_SUB_32 or PDL_LPC_SUB_64 or PDL_LPC_SUB_512 or PDL_LPC_SUB_1024 or PDL_LPC_SUB_2048 or PDL_LPC_SUB_4096 or PDL_LPC_SUB_16384 or PDL_LPC_SUB_32768 or PDL_LPC_SUB_65536 or PDL_LPC_SUB_131072 or PDL_LPC_SUB_262144 or PDL_LPC_SUB_524288	Select the oscillation settling time of the sub clock oscillator before the CPU resumes after exiting from software standby mode. When updating this value, the sub clock oscillator must be stopped.

Description (3/3)

[data4]
Select the HOCO oscillator waiting times.
If no selections are required, specify PDL_NO_DATA.

- HOCO oscillator waiting time

PDL_LPC_HOCO_40 or PDL_LPC_HOCO_72 or PDL_LPC_HOCO_104 or PDL_LPC_HOCO_136 or PDL_LPC_HOCO_180 or PDL_LPC_HOCO_200 or PDL_LPC_HOCO_232 or PDL_LPC_HOCO_264 or PDL_LPC_HOCO_296 or PDL_LPC_HOCO_328 or PDL_LPC_HOCO_360 or PDL_LPC_HOCO_392 or PDL_LPC_HOCO_424 or PDL_LPC_HOCO_456 or PDL_LPC_HOCO_488 or PDL_LPC_HOCO_520 or PDL_LPC_HOCO_3072 or PDL_LPC_HOCO_5120 or PDL_LPC_HOCO_7168 or PDL_LPC_HOCO_9216 or PDL_LPC_HOCO_11264 or PDL_LPC_HOCO_13312 or PDL_LPC_HOCO_15360 or PDL_LPC_HOCO_17408 or PDL_LPC_HOCO_19456 or PDL_LPC_HOCO_21504 or PDL_LPC_HOCO_23552 or PDL_LPC_HOCO_25600 or PDL_LPC_HOCO_27648 or PDL_LPC_HOCO_29696 or PDL_LPC_HOCO_31744 or PDL_LPC_HOCO_33792	Select the oscillation settling time of the HOCO oscillator before the CPU resumes after exiting from software standby mode. When updating this value, the HOCO oscillator must be stopped.

Return value

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

Category

LPC

References

R_LPC_Control, R_CGC_Control, R_CGC_Set

Remarks

- When operating power control mode switching is in progress, do not call this function.
- When the ROM is in program or erase mode, do not call this function.
- During the period from the time of WAIT instruction issuance for a sleep mode transition, to return from sleep mode to normal operation, do not call this function.
- Use R_CGC_Control to stop and start the clocks as required.
- When switching from normal power consumption mode to low power consumption mode, call R_CGC_Set to change the clock settings before calling this function.
- When PDL_LPC_SLEEP_RETURN_SWITCH_HOCO is selected, the frequencies of the internal clocks (ICLK, PCLKD, PCLKB and FCLK) must be no more than the selected clock source frequency ÷ 2 before a transition is made to sleep mode.
- The sleep mode return clock source switching function and clock source switching function by the ELC cannot be used at the same time.
- When HOCO is operating, low-speed operating mode 2 cannot be selected.
- When HOCO frequency is 32, 36.864 or 40 MHz, minimum waiting time is 180 cycles. When the HOCO frequency is 50 MHz, the minimum waiting time is 200 cycles.
- If PDL_LPC_SLEEP_RETURN_SWITCH_HOCO or PDL_LPC_SLEEP_RETURN_SWITCH_MAIN is selected, when the CPU is restored from the sleep mode, middle-speed operating mode A will be automatically turned on.
- When one of the frequencies of the internal clocks (ICLK, PCLKD, PCLKB and FCLK) is set to 1/1 frequency division, the PDL_LPC_SLEEP_RETURN_SWITCH_HOCO setting is prohibited.

Program example

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

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

void func( void )
{
    /* Set main clock oscillator waiting time to 32 cycles */
    R_LPC_Create(
        PDL_NO_DATA,
        PDL_LPC_MAIN_32,
        PDL_NO_DATA,
        PDL_NO_DATA
    );
}
```

2) R_LPC_Control

Synopsis

Select a low power consumption mode.

Prototype

```
bool R_LPC_Control(
    uint16_t data      // Mode selection
);
```

Description

Transition to one of the low power modes.

[data]
Control selection. All selections are optional. The default settings are shown in **bold**.
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	Select the mode to be entered.

- Operating power control

PDL_LPC_CHANGE_MIDDLE_SPEED_A or PDL_LPC_CHANGE_MIDDLE_SPEED_B or PDL_LPC_CHANGE_LOW_SPEED_1 or PDL_LPC_CHANGE_LOW_SPEED_2	Select the operating power control mode

- Sleep mode return clock source switching

PDL_LPC_SLEEP_RETURN_CHANGE_DISABLE or PDL_LPC_SLEEP_RETURN_CHANGE_HOCO or PDL_LPC_SLEEP_RETURN_CHANGE_MAIN	Control clock source switching at cancellation of sleep mode

- All-module clock stop cancellation modification

PDL_LPC_TMR_OFF or PDL_LPC_TMR_UNIT_0 or PDL_LPC_TMR_UNIT_1 or PDL_LPC_TMR_BOTH	Select whether the TMR units can be used to exit from All-module clock stop mode.

Return value

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

Category

LPC

References

R_LPC_Create

Remarks

- Sleep mode is utilised by some peripheral drivers to turn off the CPU when required.
- When entering software standby, the oscillation stop detection function is disabled. The detection is re-enabled if software standby mode is interrupted.
- If Sleep mode return clock source switching has been enabled, the only possible clock sources are the LOCO or sub-clock oscillator.
- When PDL_LPC_SLEEP_RETURN_CHANGE_HOCO is selected, the frequencies of the internal clocks (ICLK, PCLKD, PCLKB and FCLK) must be no more than the selected clock source frequency ÷ 2 before a transition is made to sleep mode.
- The sleep mode return clock source switching function and clock source switching function by the ELC cannot be used at the same time.
- Do not set up the DMAC and DTC to rewrite any registers related to WDT while the chip is in sleep mode.
- If IWDT is stopped, do not set up the DMAC and DTC to rewrite any registers related to IWDT while the chip is in sleep mode.
- If a condition for the independent watchdog timer to stop counting applied at the time of a transition to all module clock stop mode, using a reset from the independent watchdog timer to release the chip from all module clock stop mode is impossible because the independent watchdog timer is stopped.
- 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.
- When one of the frequencies of the internal clocks (ICLK, PCLKD, PCLKB and FCLK) is set to 1/1 frequency division, the PDL_LPC_SLEEP_RETURN_CHANGE_HOCO setting is prohibited.
- In all module clock stop mode while the POE interrupt is enabled, the operation is not restored by POE interrupt occurance. However, the POE interrupt occurs after the operation is restored by another source.
- Do not try to change operating mode if ROM Program Erase mode is set or a mode transition is already in progress. This function will return false if this is detected.

Program example

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

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

void func( void )
{
    /* Enter software standby mode */
    R_LPC_Control(
        PDL_LPC_MODE_SOFTWARE_STANDBY
    );
}
```

3) R_LPC_GetStatus

Synopsis

Read the status flags.

Prototype

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

Description

Read the Low power status flags.

[data]
The status flags shall be stored in the format below.

b15 – b9	b8
0	Operating Power Control Mode transition flag 0: Transition completed 1:During Transition

b7 – b4	b3	b2	b1	b0
Event detection flags (0: not detected; 1: detected)				
0	LVD2	LVD1	LVD0	Power-on reset

Return value

True.

Category

LPC

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

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

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

void func( void )
{
    uint16_t status_flags;

    /* Read the low power status flag */
    R_LPC_GetStatus(
        &status_flags
    );
}
```

4.2.9. Register Write Protection

1) R_RWP_Control

Synopsis

Control register write protection.

Prototype

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

Description

Control register write protection.

[data]
Write enable control
To set multiple options at the same time, use "|" to separate each value.

- Register write control

PDL_RWP_ENABLE_CGC_WRITE or PDL_RWP_DISABLE_CGC_WRITE	Enable or disable writing to CGC registers.
PDL_RWP_ENABLE_MODE_RESET_WRITE or PDL_RWP_DISABLE_MODE_RESET_WRITE	Enable or disable writing to Mode and Reset registers.
PDL_RWP_ENABLE_LVD_WRITE or PDL_RWP_DISABLE_LVD_WRITE	Enable or disable writing to LVD registers.
PDL_RWP_ENABLE_MPC_WRITE or PDL_RWP_DISABLE_MPC_WRITE	Enable or disable MPC Register access.

Return value

True if the parameter is valid; otherwise false.

Category

RWP

References

None.

Remarks

- To allow for nested function calls, the access to the enabling / disabling of register protection is done using a reference counting method. Hence a call to disable a register access may only decrement a reference counter and not actually apply the write protection.
- Other RPDL functions automatically enable and disable access to registers as required so this function is normally not required.

Program example

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

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

void func( void )
{
    /* Enable access to the LVD registers */
    R_RWP_Control(
        PDL_RWP_ENABLE_LVD_WRITE
    );
}
```

2) R_RWP_GetStatus

Synopsis

Get the status of the register protection.

Prototype

```
bool R_RWP_GetStatus(
        uint8_t * data1,    // Status flags pointer
        uint8_t * data2     // Status flags pointer
);
```

Description

Get the status of the register protection.

[data1]
The Protect Register (PRCR). If the value is not required, specify PDL_NO_PTR.

b7 – b4	b3	b2	b1	b0
0	LVD 0: Write Disabled 1: Write Enabled	0	Mode and Reset 0: Write Disabled 1: Write Enabled	CGC 0: Write Disabled 1: Write Enabled

[data2]
The MPC Write Protect Register (PWPR). If the value is not required, specify PDL_NO_PTR.

b7	b6	b5 - b0
BOWI 0: Writing to the PFSWE bit is enabled 1: Writing to the PFSWE bit is disabled	PFSWE 0: Writing to the PFS register is disabled 1: Writing to the PFS register is enabled	0

Return value

True.

Category

RWP

Reference

None

Remarks

Program example

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

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

void func(void)
{
    uint8_t PRCR_value;
    uint8_t PWPR_value;

    /* Read the protection registers */
    R_RWP_GetStatus(
         &PRCR_value,
         &PWPR_value
    );
}
```

4.2.10. Bus Controller

1) R_BSC_Set

Synopsis

Configure the internal bus operation.

Prototype

```
bool R_BSC_Set(
        uint16_t data     // Bus priority selection
);
```

Description

Configure the priority of the internal and external buses.

[data]

- Bus priority control. If multiple selections are required, use "|" to separate each selection. The default settings are shown in **bold**.

	Bus to be accessed	Priority
PDL_BSC_PRIORITY_RAM_MB2 or PDL_BSC_PRIORITY_RAM_CPU	RAM	Fixed to internal main bus 2, or toggled with the CPU bus.
PDL_BSC_PRIORITY_ROM_MB2 or PDL_BSC_PRIORITY_ROM_CPU	ROM	
PDL_BSC_PRIORITY_PB1_MB2 or PDL_BSC_PRIORITY_PB1_MB1	Peripheral 1	Fixed to internal main bus 2, or toggled with internal main bus 1.
PDL_BSC_PRIORITY_PB2_MB2 or PDL_BSC_PRIORITY_PB2_MB1	Peripheral 2	
PDL_BSC_PRIORITY_PB6_MB2 or PDL_BSC_PRIORITY_PB6_MB1	Peripheral 6	

Return value

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

Category

Bus Controller

Reference

None.

Remarks

- If it is necessary to call this function, call it once only. Ensure that both the DTC and DMAC are stopped.

Program example

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

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

void func(void)
{
    /* Toggle the priority to the Internal Peripheral Bus 1 between
        Main Bus 1 and Main Bus 2. */
    R_BSC_Set(
        PDL_BSC_PRIORITY_PB1_MB1
    );
}
```

2) R_BSC_Create

Synopsis

Configure the BSC error detection.

Prototype

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

Description (1/1)

Configure the error detection and register the callback function

[data1]

- Error monitoring

PDL_BSC_ERROR_ILLEGAL_ADDRESS_DISABLE or PDL_BSC_ERROR_ILLEGAL_ADDRESS_ENABLE	Disable or enable 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_Set, R_BSC_Control

Remarks

- If required, call R_BSC_Set before using this function.
- 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 illegal address detection and register a callback. */
    R_BSC_Create(
        PDL_BSC_ERROR_ILLEGAL_ADDRESS_ENABLE,
        BusErrorFunc,
        5
    );
}
```

3) R_BSC_Control

Synopsis

Modify the Bus Controller operation.

Prototype

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

Description

Control the BSC operation

[data]
Control the BSC operation.

- Error clearing

PDL_BSC_ERROR_CLEAR	Clear the bus-error status registers.

- Disable bus error interrupt request

PDL_BSC_DISABLE_BUSERR_IRQ	Disable bus error interrupt requests.

Return value

True

Category

Bus Controller

Reference

R_BSC_Create

Remarks

- This function can be called from the error handling function (assigned in R_BSC_Create).
- This function will clear the Interrupt Status Flag indirectly.

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
    );
}
```

4) R_BSC_GetStatus

Synopsis

Read the External Bus Controller status registers.

Prototype

```
bool R_BSC_GetStatus(
        uint8_t * data1,     // The status register 1 storage location
        uint16_t * data2     // The status register 2 storage location
);
```

Description

Read the BSC status registers

[data1]
The status flags shall be stored according to register BERSR1 format as below.
Specify PDL_NO_PTR if this information is not required.

b7	b6 – b4	b3 – b1	b0
0	The bus master that caused the error 000 : CPU 011 : DTC or DMAC	0	Illegal address access 0: None 1: Occurred

[data2]
The status flags shall be stored according to register BERSR2 format as below.
Specify PDL_NO_PTR if this information is not required.

b15 – b3	b2 – b0
The upper 13 bits of the address that was accessed when the bus error occurred (in units of 512 Kbytes).	0

Return value

True.

Category

Bus Controller

Reference

R_BSC_Control

Remarks

- Call R_BSC_Control to clear the status registers after reading the status.

Program example

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

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

void func(void)
{
    uint8_t status1;
    uint16_t status2;

    /* Read the flags */
    R_BSC_GetStatus(
        &status1,
        &status2
    );
}
```

4.2.11. DMA Controller

1) R_DMAC_Create

Synopsis

Configure the DMA controller.

Prototype

```
bool R_DMAC_Create(
        uint8_t data1,      // Channel selection
        uint32_t data2,     // Configuration selection
        uint8_t data3,      // Trigger selection
        void * data4,       // Source start address
        void * data5,       // Destination start address
        uint16_t data6,     // Transfer count
        uint16_t data7,     // Repeat or Block size
        int32_t data8,      // Address offset
        uint32_t data9,     // Source address extended repeat area
        uint32_t data10,    // Destination address extended repeat area
        void * func,        // Callback function
        uint8_t data11      // Interrupt priority level
);
```

Description (1/3)

Set up a DMA channel.

[data1]
The channel number n (where n = 0 to 3).

[data2]
Configure the operation of channel DMAn.
If multiple selections are required, use "|" to separate each selection.
The default settings are shown in **bold**.

- Transfer mode selection

PDL_DMAC_NORMAL or PDL_DMAC_REPEAT or PDL_DMAC_BLOCK	Normal or Repeat or Block mode.
PDL_DMAC_SOURCE or PDL_DMAC_DESTINATION	If Repeat or Block mode is selected, the source or destination side can be selected as the Repeat or Block area. This selection is optional.

- Address direction selection

PDL_DMAC_SOURCE_ADDRESS_FIXED or PDL_DMAC_SOURCE_ADDRESS_PLUS or PDL_DMAC_SOURCE_ADDRESS_MINUS or PDL_DMAC_SOURCE_ADDRESS_OFFSET	Leave the source address unchanged, increment it, decrement it or modify it by the value specified in parameter data8. Address offset is valid only for n = 0.
PDL_DMAC_DESTINATION_ADDRESS_FIXED or PDL_DMAC_DESTINATION_ADDRESS_PLUS or PDL_DMAC_DESTINATION_ADDRESS_MINUS or PDL_DMAC_DESTINATION_ADDRESS_OFFSET	Leave the destination address unchanged, increment it, decrement it or modify it by the value specified in parameter data8. Address offset is valid only for n = 0.

- Transfer data size

PDL_DMAC_SIZE_8 or PDL_DMAC_SIZE_16 or PDL_DMAC_SIZE_32	Select 8, 16 or 32 bits for the data to be transferred.

- Interrupt generation (optional).

PDL_DMAC_IRQ_END	Transfer completion.
PDL_DMAC_IRQ_ESCAPE_END	Escape end.
PDL_DMAC_IRQ_REPEAT_SIZE_END	1-repeat size or 1-block data transfer completion.
PDL_DMAC_IRQ_EXT_SOURCE	Extended repeat area overflow on the source.
PDL_DMAC_IRQ_EXT_DESTINATION	Extended repeat area overflow on the destination.

Description (2/3)

- Start trigger forwarding

PDL_DMAC_TRIGGER_CLEAR or PDL_DMAC_TRIGGER_FORWARD	When the DMAC transfer is complete, clear the DMAC activation trigger or pass it on to the CPU.

- DTC trigger control

PDL_DMAC_DTC_TRIGGER_DISABLE or PDL_DMAC_DTC_TRIGGER_ENABLE	Disable or enable activation of the DTC when an event specified in the “Interrupt generation” options occurs.

[data3]
Select one activation source for channel DMAn.

- Trigger selection

Name	Trigger cause
PDL_DMAC_TRIGGER_SW or	By software.
PDL_DMAC_TRIGGER_CMT0 or	Compare match on channel CMTn (n = 0 to 3).
PDL_DMAC_TRIGGER_CMT1 or	
PDL_DMAC_TRIGGER_CMT2 or	
PDL_DMAC_TRIGGER_CMT3 or	
PDL_DMAC_TRIGGER_SPI0_RX or	Receive buffer full on RSPI channel 0.
PDL_DMAC_TRIGGER_SPI0_TX or	Transmit buffer empty on RSPI channel 0.
PDL_DMAC_TRIGGER_IRQ0 or	Valid edge detected on pin IRQn (n = 0 to 3).
PDL_DMAC_TRIGGER_IRQ1 or	
PDL_DMAC_TRIGGER_IRQ2 or	
PDL_DMAC_TRIGGER_IRQ3 or	
PDL_DMAC_TRIGGER_ADC12 or	Conversion completed on the 12-bit ADC unit.
PDL_DMAC_TRIGGER_ADC12_GBADI or	Conversion completed on group B of the 12-bit ADC unit.
PDL_DMAC_TRIGGER_ELSR18I or	Event link interrupt
PDL_DMAC_TRIGGER_MTU0 or	Input capture or compare match on MTU channel n (n = 0 to 4).
PDL_DMAC_TRIGGER_MTU1 or	
PDL_DMAC_TRIGGER_MTU2 or	
PDL_DMAC_TRIGGER_MTU3 or	
PDL_DMAC_TRIGGER_MTU4 or	
PDL_DMAC_TRIGGER_SCI1_RX or	Receive buffer full on SCI unit n (n = 1, 5, 6, 9 or 12).
PDL_DMAC_TRIGGER_SCI5_RX or	
PDL_DMAC_TRIGGER_SCI6_RX or	
PDL_DMAC_TRIGGER_SCI9_RX or	
PDL_DMAC_TRIGGER_SCI12_RX or	
PDL_DMAC_TRIGGER_SCI1_TX or	Transmit buffer empty on SCI unit n (n = 1, 5, 6, 9 or 12).
PDL_DMAC_TRIGGER_SCI5_TX or	
PDL_DMAC_TRIGGER_SCI6_TX or	
PDL_DMAC_TRIGGER_SCI9_TX or	
PDL_DMAC_TRIGGER_SCI12_TX or	
PDL_DMAC_TRIGGER_IIC0_RX or	Receive buffer full on I²C channel 0.
PDL_DMAC_TRIGGER_IIC0_TX	Transmit buffer empty on I²C channel 0.

[data4]
The source start address.

[data5]
The destination start address.

[data6]
The number of transfers to take place.
For normal mode: valid between 0 and 65535 (0 = free running mode).
For repeat and block mode: valid between 0 and 1023 (0 = 1024 transfers).

Description (2/3)

[data7]
The repeat or block size for each transfer.
For repeat and block mode: valid between 0 and 1023 (0 = 1024 units).
Ignored in normal mode.

[data8]
The address offset value. The range is from +16,777,215 to -16,777,216.
This value is ignored if the offset function is not selected.

[data9]
The source address extended repeat value. The value can be any power of 2, from 2^1 to 2^{27}.
Specify PDL_NO_DATA if the extended repeat function is not required for the source address.

[data10]
The destination address extended repeat value.
The value can be any power of 2, from 2^1 to 2^{27}.
Specify PDL_NO_DATA if the extended repeat function is not required for the destination address.

[**func**]
The function to be called when a DMA transfer completes.
Specify PDL_NO_FUNC if not required.

[data11]
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

DMA controller

Reference

None.

Remarks

- If another peripheral will be used to trigger a DMA transfer, call this function before calling the Create function for the peripheral.
- Some peripheral channels are not available on some device packages. Please check the hardware manual.
- 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_dmac.h"

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

void func(void)
{
    /* Configure DMA channel 2 */
    R_DMAC_Create(
        2,
        PDL_DMAC_NORMAL | \
        PDL_DMAC_SOURCE_ADDRESS_PLUS | \
        PDL_DMAC_DESTINATION_ADDRESS_PLUS | \
        PDL_DMAC_SIZE_8,
        PDL_DMAC_TRIGGER_IRQ0,
        (void*) 0x0000AA00,
        (void*) 0x0000BB00,
        10,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_FUNC,
        0
    );
}
```

2) R_DMAC_Destroy

Synopsis

Disable the DMA controller.

Prototype

```
bool R_DMAC_Destroy(
        uint8_t data      // Channel number
);
```

Description

Shutdown the DMAC module.

[data]
The channel number n (where n = 0 to 3).

Return value

True if the shutdown succeeded; otherwise false.

Category

DMA controller

Reference

R_DMAC_Create.

Remarks

- If all channels have been suspended, the DMAC module will be shut down.
- Disabling the DMAC module will also shut down the DTC.
- If another peripheral is being used to trigger a DMA transfer, stop the triggers from that peripheral (using Control or Destroy for that peripheral) before calling this function.

Program example

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

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

void func(void)
{
    /* Shutdown DMAC channel 2 */
    R_DMAC_Destroy(
        2
    );
}
```

3) R_DMAC_Control

Synopsis

Control the DMA controller.

Prototype

```
bool R_DMAC_Control (
        uint8_t data1,      // Channel number
        uint16_t data2,     // Control options
        void * data3,       // Source start address
        void * data4,       // Destination start address
        uint16_t data5,     // Transfer count
        uint16_t data6,     // Repeat or Block size
        int32_t data7,      // Address offset
        uint32_t data8,     // Source address extended repeat area
        uint32_t data9      // Destination address extended repeat area
);
```

Description (1/2)

Change the state of a DMA controller channel.

[data1]
The channel number n (where n = 0 to 3).

[data2]
Control the channel operation.
If multiple selections are required, use "|" to separate each selection.

- Enable / suspend control

PDL_DMAC_ENABLE	Enable / re-enable DMA transfers.
PDL_DMAC_SUSPEND	Suspend DMA transfers.

- Software trigger control

PDL_DMAC_START or PDL_DMAC_START_RUN or PDL_DMAC_STOP	Start a DMA transfer. Start DMA transfers until stopped. Stop software-triggered transfers.

- Transfer end interrupt flag control

PDL_DMAC_CLEAR_DTIF	Clear the Transfer End flag.
PDL_DMAC_CLEAR_ESIF	Clear the Transfer Escape End flag.

- The values to be modified.

PDL_DMAC_UPDATE_SOURCE	Source address, using parameter data3.
PDL_DMAC_UPDATE_DESTINATION	Destination address, using parameter data4.
PDL_DMAC_UPDATE_COUNT	Transfer count, using parameter data5.
PDL_DMAC_UPDATE_SIZE	Repeat or Block size, using parameter data6.
PDL_DMAC_UPDATE_OFFSET	Address offset, using parameter data7.
PDL_DMAC_UPDATE_REPEAT_SOURCE	Source address extended repeat area, using parameter data8.
PDL_DMAC_UPDATE_REPEAT_DESTINATION	Destination address extended repeat area, using parameter data9.

[data3]
The new source address. Specify PDL_NO_PTR if not required.

[data4]
The new destination address. Specify PDL_NO_PTR if not required.

[data5]
The transfer count value. Specify PDL_NO_DATA if not required.

Description (2/2)

[data6]
The repeat or block size for each transfer.
Valid between 0 and 1023 (0 = 1024 units).
Ignored in normal mode.
Specify PDL_NO_DATA if not required.

[data7]
The address offset value.
The range is from +16,777,215 to -16,777,216.
This value is ignored if the offset function is not selected.
Specify PDL_NO_DATA if not required.

[data8]
The source address extended repeat value.
The value can be any power of 2, from 2^1 to 2^{27}.
Specify PDL_NO_DATA if not required.

[data9]
The destination address extended repeat value.
The value can be any power of 2, from 2^1 to 2^{27}.
Specify PDL_NO_DATA if not required.

Return value

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

Category

DMA controller

Reference

R_DMAC_Create

Remarks

- The Software trigger control is valid only if the Software trigger option has been selected.
- This function must be called in order to start the DMAC.
- The Suspend / Enable and Start control is executed at the end of the function. If a channel has completed a transfer, parameters may be changed and the channel re-enabled in one function call.

Program example

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

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

#include <string.h>

const char source_string_1[]="Renesas RX220";
volatile char destination_string_1[]=".....................";

void func(void)
{
    /* Re-enable transfers on channel 2 */
    R_DMAC_Control(
        2,
        PDL_DMAC_ENABLE,
        PDL_NO_PTR,
        PDL_NO_PTR,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA
    );

    /* Reload and trigger channel 1 */
    R_DMAC_Control(
        1,
        PDL_DMAC_ENABLE | PDL_DMAC_START |
         PDL_DMAC_UPDATE_SOURCE |
          PDL_DMAC_UPDATE_DESTINATION |
         PDL_DMAC_UPDATE_COUNT |
         PDL_DMAC_UPDATE_SIZE,
        source_string_1,
        destination_string_1,
        1,
        (uint16_t)strlen(source_string_1),
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA
    );
}
```

4) R_DMAC_GetStatus

Synopsis

Check the status of a DMA channel.

Prototype

```
bool R_DMAC_GetStatus(
        uint8_t data1,      // Channel number
        uint8_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
        uint16_t * data6    // Current Repeat or Block size count pointer
);
```

Description

Return status flags and current channel registers.

[data1]
The channel number n (where n = 0 to 3).

[data2]
The status flags shall be stored in the following format.
Specify PDL_NO_PTR if the flags are not to be read.

b7 – b5	b4	b3	b2	b1	b0
0	Interrupt request (IR)	Transfer Escape End interrupt (ESIF)	Transfer End interrupt (DTIF)	Status (ACT)	Transfer enable (DTE)
		0: Idle 1: Generated	0: Idle 1: Generated	0: Idle 1: Operating	0: Disabled 1: Enabled

[data3]
Where the current source address shall be stored. Specify PDL_NO_PTR if it is not required.

[data4]
Where the current destination address shall be stored. Specify PDL_NO_PTR if it is not required.

[data5]
Where the current transfer count shall be stored. Specify PDL_NO_PTR if it is not required.

[data6]
Where the current repeat or block size count shall be stored. Specify PDL_NO_PTR if it is not required.

Return value

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

Category

DMA controller

Reference

R_DMAC_Create

Remarks

- If the Interrupt request flag is set to 1, the flag will be cleared to 0 by this function.

Program example

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

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

void func(void)
{
    uint8_t StatusValue;
    uint32_t SourceAddr;

    /* Read the status and current source address for channel 2 */
    R_DMAC_GetStatus(
        2,
        &StatusValue,
        &SourceAddr,
        PDL_NO_PTR,
        PDL_NO_PTR,
        PDL_NO_PTR
    );
}
```

4.2.12. 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

PDL_DTC_READ_SKIP_DISABLE or PDL_DTC_READ_SKIP_ENABLE	Disable or enable skipping of transfer data read when the vector numbers match.

- Address size control

PDL_DTC_ADDRESS_FULL 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 1024 bytes boundary.

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.

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 = 0x00000400
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

PDL_DTC_CHAIN_DISABLE or PDL_DTC_CHAIN_CONTINUOUS or PDL_DTC_CHAIN_0	Disable chain transfer operation, Perform continuous chain transfers or Perform a chain transfer when the transfer counter is changed from 1 to 0, or 1 to transfer size / block size.

- Interrupt generation

PDL_DTC_IRQ_COMPLETE 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	Compare match on channel CMTn (n = 0 to 3).
PDL_DTC_TRIGGER_CMT1 or	
PDL_DTC_TRIGGER_CMT2 or	
PDL_DTC_TRIGGER_CMT3 or	
PDL_DTC_TRIGGER_SPI0_RX or	Receive buffer full on SPI channel 0.
PDL_DTC_TRIGGER_SPI0_TX or	Transmit buffer empty on SPI 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_ADC12 or	Conversion completed on the 12-bit ADC unit.
PDL_DTC_TRIGGER_ADC12_GBADI or	Conversion completed on group B of the 12-bit ADC unit.
PDL_DTC_TRIGGER_ELSR18I or	Event link interrupt
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	Compare match or input capture A on MTU channel n (n = 0 to 4).
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	Compare match or input capture B on MTU channel n (n = 0 to 4).
PDL_DTC_TRIGGER_TGIC0 or PDL_DTC_TRIGGER_TGIC3 or PDL_DTC_TRIGGER_TGIC4 or	Compare match or input capture C on MTU channel n (n = 0, 3 or 4).
PDL_DTC_TRIGGER_TGID0 or PDL_DTC_TRIGGER_TGID3 or PDL_DTC_TRIGGER_TGID4 or	Compare match or input capture D on MTU channel n (n = 0, 3 or 4).
PDL_DTC_TRIGGER_TGIU5 or	Compare match or input capture U on MTU channel 5.
PDL_DTC_TRIGGER_TGIV5 or	Compare match or input capture V on MTU channel 5.
PDL_DTC_TRIGGER_TGIW5 or	Compare match or input capture W on MTU channel 5.
PDL_DTC_TRIGGER_TCIV4 or	Counter over or underflow on MTU channel 4.
PDL_DTC_TRIGGER_CMIA0 or PDL_DTC_TRIGGER_CMIA1 or PDL_DTC_TRIGGER_CMIA2 or PDL_DTC_TRIGGER_CMIA3 or	Compare match A on TMR channel n (n = 0 to 3).
PDL_DTC_TRIGGER_CMIB0 or PDL_DTC_TRIGGER_CMIB1 or PDL_DTC_TRIGGER_CMIB2 or PDL_DTC_TRIGGER_CMIB3 or	Compare match B on TMR channel n (n = 0 to 3).
PDL_DTC_TRIGGER_DMACI0 or PDL_DTC_TRIGGER_DMACI1 or PDL_DTC_TRIGGER_DMACI2 or PDL_DTC_TRIGGER_DMACI3 or	Transfer complete on DMAC channel n (n = 0 to 3).
PDL_DTC_TRIGGER_RXI1 or PDL_DTC_TRIGGER_RXI5 or PDL_DTC_TRIGGER_RXI6 or PDL_DTC_TRIGGER_RXI9 or PDL_DTC_TRIGGER_RXI12 or	Receive buffer full on SCI channel n (n = 1, 5, 6, 9 or 12).
PDL_DTC_TRIGGER_TXI1 or PDL_DTC_TRIGGER_TXI5 or PDL_DTC_TRIGGER_TXI6 or PDL_DTC_TRIGGER_TXI9 or PDL_DTC_TRIGGER_TXI12 or	Transmit buffer empty on SCI channel n (n = 1, 5, 6, 9 or 12).
PDL_DTC_TRIGGER_IIC0_RX or	Receive buffer full on I²C channel 0.
PDL_DTC_TRIGGER_IIC0_TX	Transmit buffer empty on I²C channel 0.

Description (3/3)

[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 size of each block transfer. Valid between 0 and 255 (0 = 256 units).
Ignored in normal or repeat mode.

Return value

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

Category

Data Transfer Controller

Reference

R_DTC_Set, R_DTC_Control

Remarks

- If address increment or decrement is selected, the address changes according to the number of bytes (1, 2 or 4) in each transfer.
- Before calling this function, call R_DTC_Set.
- Call this function before configuring the peripherals that will be involved in the data transfer.
- Call this function once for each peripheral that will trigger a transfer, and for each chained transfer.
- For chain transfers, each transfer data area in the chain must be contiguous.
- When all calls to this function are complete, call R_DTC_Control to start the DTC.

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,
        (void*)0x0000AA00,
        (void*)0x0000BB00,
        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	• This function will also shut down the DMAC. • 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. iii. Stop the DMAC.

Program example

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

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

void func(void)
{
    /* Shutdown the DTC (& DMAC) */
    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.
If multiple selections are required, use "|" to separate each selection.

- 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 0 and 255 (0 = 256 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).
- Start the DTC before generating a transfer trigger.

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,
        (void*)0x0000BB00,
        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 bit 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
    );
}
```

4.2.13. Event Link Controller

1) R_ELC_Create

Synopsis

Enable the ELC module.

Prototype

```
bool R_ELC_Create(
    void* func1,        // ELC Interrupt1 callback function.
    uint8_t data1,      // ELC Interrupt1 Interrupt priority level.
);
```

Description

Enable the ELC module and provide callback function registration.

[func1]
The function to be called when an ELC Interrupt1 (ELSR18I) occurs.
Specify PDL_NO_FUNC if not required.

[data1]
The interrupt priority level for Interrupt1.
If using Interrupt1 select between 1 (lowest priority) and 15 (highest priority) otherwise set to 0.

Return value

True if all parameters are valid; otherwise false.

Category

Event Link Controller

Reference

R_ELC_Control

Remarks

- Call this function before using any other ELC function.
- If using ELC Interrupts, use this function to register the callback functions and R_ELC_Control to create a link.

Program example

```
#include "r_pdl_elc.h"

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

void Interrupt1_CallBack(void);

void func(void)
{
    /* Enable the module and setup an Interrupt1 callback. */
    R_ELC_Create(
        Interrupt1_CallBack,
        5
    );
}
```

2) R_ELC_Destroy

Synopsis	Disable the ELC module.
Prototype	**bool R_ELC_Destroy(** **void** // No parameter is required **);**
Description	Disable all links and enable the ELC module stop state.
Return value	True
Category	Event Link Controller
Reference	R_ELC_Create
Remarks	None
Program example	

```
#include "r_pdl_elc.h"

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

void func(void)
{
    R_ELC_Destroy();
}
```

3) R_ELC_Read

Synopsis

Read the ELC port buffer.

Prototype

```
bool R_ELC_Read
    uint8_t data1,      // Port
    uint8_t*data2       // Storage
);
```

Description

Read the ELC port buffer for the specified port.

[data1]

- Port selection

PDL_ELC_PORT_B	Select the port whose buffer should be read.

[data2]

Address where the port buffer value will be stored.

Return value

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

Category

Event Link Controller

Reference

R_ELC_Create, R_ELC_Control

Remarks

None

Program example

```
#include "r_pdl_elc.h"

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

void func(void)
{
    uint8_t value;

    R_ELC_Read(
        PDL_ELC_PORT_B,
        &value
    );
}
```

4) R_ELC_Write

Synopsis

Write to the ELC port buffer.

Prototype

```
bool R_ELC_Write
    uint8_t data1,      // Port
    uint8_t data2       // Value
);
```

Description

Write to the ELC port buffer of the specified port.

[data1]

- Port selection

PDL_ELC_PORT_B	Select the port whose buffer should be written to.

[data2]

Value to write.

Return value

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

Category

Event Link Controller

Reference

R_ELC_Create, R_ELC_Control

Remarks

- If an event occurs while updating a port with bit rotation enabled, abnormal operation may occur.

Program example

```
#include "r_pdl_elc.h"

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

void func(void)
{
    R_ELC_Write(
        PDL_ELC_PORT_B,
        0xAA
    );
}
```

5) R_ELC_Control

Synopsis

Control the ELC.

Prototype

```
bool R_ELC_Control
    uint16_t data1,     // Operation to perform
    uint32_t data2,     // Operation-specific configuration
    uint32_t data3      // Operation-specific configuration
);
```

Description (1/5)

Setup and then control the ELC.

[data1]

- Operation selection

PDL_ELC_ENABLE or	Enable the linkage of all configured events.
PDL_ELC_DISABLE or	Disable the linkage of all events.
PDL_ELC_CREATE_LINK or	Create a link between an event and a module. This function must be called for each link required.
PDL_ELC_REMOVE_LINK or	Remove a link previously created.
PDL_ELC_TIMER_OPERATION or	Configure the operation of a timer when triggered by an event.
PDL_ELC_PORT_GROUP or	Define a port group by selecting the pins within a port that make up the group.
PDL_ELC_PORT_CONTROL or	Input and output control options for a port group.
PDL_ELC_SINGLE_PORT or	Selection of a single port and configuration of it when used as an event generator and / or a triggered module.
PDL_ELC_SOFTWARE_EVENT or	Generate a software event.
PDL_ELC_TRIGGER	Set DTC trigger from ELC Interrupts options.

NOTE: The specification of parameters data2 and data3 depends on the operation specified in parameter data1. Hence see the section below relating to the specific operation required.

Operation (data1) = PDL_ELC_ENABLE

[data2] Not used. Specify PDL_NO_DATA
[data3] Not used. Specify PDL_NO_DATA

Operation (data1) = PDL_ELC_DISABLE

[data2] Not used. Specify PDL_NO_DATA
[data3] Not used. Specify PDL_NO_DATA

Operation (data1) = PDL_ELC_CREATE_LINK

[data2]

- Module Selection

PDL_ELC_LINK_MODULE_MTU2_CHANNEL_1 or PDL_ELC_LINK_MODULE_MTU2_CHANNEL_2 or PDL_ELC_LINK_MODULE_MTU2_CHANNEL_3 or PDL_ELC_LINK_MODULE_MTU2_CHANNEL_4 or PDL_ELC_LINK_MODULE_TMR_CHANNEL_0 or PDL_ELC_LINK_MODULE_TMR_CHANNEL_2 or PDL_ELC_LINK_MODULE_ADC12 or PDL_ELC_LINK_MODULE_INTERRUPT_1 or PDL_ELC_LINK_MODULE_OUTPUT_PORT_B_GROUP or PDL_ELC_LINK_MODULE_INPUT_PORT_B_GROUP or PDL_ELC_LINK_MODULE_SINGLE_PORT_0 or PDL_ELC_LINK_MODULE_SINGLE_PORT_1 or	Select the module that will be triggered by the event.

Description (2/5)

[data3]

- Event selection.

PDL_ELC_LINK_EVENT_MTU2_CHANNEL_1_COMPARE_MATCH_1A or PDL_ELC_LINK_EVENT_MTU2_CHANNEL_1_COMPARE_MATCH_1B or PDL_ELC_LINK_EVENT_MTU2_CHANNEL_1_OVERFLOW or PDL_ELC_LINK_EVENT_MTU2_CHANNEL_1_UNDERFLOW or PDL_ELC_LINK_EVENT_MTU2_CHANNEL_2_COMPARE_MATCH_2A or PDL_ELC_LINK_EVENT_MTU2_CHANNEL_2_COMPARE_MATCH_2B or PDL_ELC_LINK_EVENT_MTU2_CHANNEL_2_OVERFLOW or PDL_ELC_LINK_EVENT_MTU2_CHANNEL_2_UNDERFLOW or PDL_ELC_LINK_EVENT_MTU2_CHANNEL_3_COMPARE_MATCH_3A or PDL_ELC_LINK_EVENT_MTU2_CHANNEL_3_COMPARE_MATCH_3B or PDL_ELC_LINK_EVENT_MTU2_CHANNEL_3_COMPARE_MATCH_3C or PDL_ELC_LINK_EVENT_MTU2_CHANNEL_3_COMPARE_MATCH_3D or PDL_ELC_LINK_EVENT_MTU2_CHANNEL_3_OVERFLOW or PDL_ELC_LINK_EVENT_MTU2_CHANNEL_4_COMPARE_MATCH_4A or PDL_ELC_LINK_EVENT_MTU2_CHANNEL_4_COMPARE_MATCH_4B or PDL_ELC_LINK_EVENT_MTU2_CHANNEL_4_COMPARE_MATCH_4C or PDL_ELC_LINK_EVENT_MTU2_CHANNEL_4_COMPARE_MATCH_4D or PDL_ELC_LINK_EVENT_MTU2_CHANNEL_4_OVERFLOW or PDL_ELC_LINK_EVENT_MTU2_CHANNEL_4_UNDERFLOW or PDL_ELC_LINK_EVENT_TMR_CHANNEL_0_COMPARE_MATCH_A0 or PDL_ELC_LINK_EVENT_TMR_CHANNEL_0_COMPARE_MATCH_B0 or PDL_ELC_LINK_EVENT_TMR_CHANNEL_0_OVERFLOW or PDL_ELC_LINK_EVENT_TMR_CHANNEL_2_COMPARE_MATCH_A2 or PDL_ELC_LINK_EVENT_TMR_CHANNEL_2_COMPARE_MATCH_B2 or PDL_ELC_LINK_EVENT_TMR_CHANNEL_2_OVERFLOW or PDL_ELC_LINK_EVENT_SCI5_ERROR or PDL_ELC_LINK_EVENT_SCI5_RECEIVE_DATA_FULL or PDL_ELC_LINK_EVENT_SCI5_TRANSMIT_DATA_EMPTY or PDL_ELC_LINK_EVENT_SCI5_TRANSMIT_END or PDL_ELC_LINK_EVENT_IIC_ERROR_OR_EVENT or PDL_ELC_LINK_EVENT_IIC_RECEIVE_DATA_FULL or PDL_ELC_LINK_EVENT_IIC_TRANSMIT_DATA_EMPTY or PDL_ELC_LINK_EVENT_IIC_TRANSMIT_END or PDL_ELC_LINK_EVENT_SPI_ERROR or PDL_ELC_LINK_EVENT_SPI_IDLE or PDL_ELC_LINK_EVENT_SPI_RECEIVE_DATA_FULL or PDL_ELC_LINK_EVENT_SPI_TRANSMIT_DATA_EMPTY or PDL_ELC_LINK_EVENT_SPI_TRANSMIT_END or PDL_ELC_LINK_EVENT_ADC12_CONVERSION_END or PDL_ELC_LINK_EVENT_LVD1_VOLTAGE_DETECTION or PDL_ELC_LINK_EVENT_DTC_TRANSFER_END or or PDL_ELC_LINK_EVENT_INPUT_PORT_GROUP_B or PDL_ELC_LINK_EVENT_SINGLE_INPUT_PORT_0 or PDL_ELC_LINK_EVENT_SINGLE_INPUT_PORT_1 or PDL_ELC_LINK_EVENT_SOFTWARE_EVENT or PDL_ELC_LINK_EVENT_DOC	Select the event that will trigger the module.

Description (3/5)

Operation (data1) = PDL_ELC_REMOVE_LINK

[data2]

- Module Selection

PDL_ELC_LINK_MODULE_MTU2_CHANNEL_1 or PDL_ELC_LINK_MODULE_MTU2_CHANNEL_2 or PDL_ELC_LINK_MODULE_MTU2_CHANNEL_3 or PDL_ELC_LINK_MODULE_MTU2_CHANNEL_4 or PDL_ELC_LINK_MODULE_TMR_CHANNEL_0 or PDL_ELC_LINK_MODULE_TMR_CHANNEL_2 or PDL_ELC_LINK_MODULE_ADC12 or PDL_ELC_LINK_MODULE_INTERRUPT_1 or PDL_ELC_LINK_MODULE_OUTPUT_PORT_B_GROUP or PDL_ELC_LINK_MODULE_INPUT_PORT_B_GROUP or PDL_ELC_LINK_MODULE_SINGLE_PORT_0 or PDL_ELC_LINK_MODULE_SINGLE_PORT_1 or	Select the module to remove the link from.

Operation (data1) = PDL_ELC_TIMER_OPERATION

[data2]

- Timer selection

PDL_ELC_TIMER_MTU2_CHANNEL_1 or PDL_ELC_TIMER_MTU2_CHANNEL_2 or PDL_ELC_TIMER_MTU2_CHANNEL_3 or PDL_ELC_TIMER_MTU2_CHANNEL_4 or PDL_ELC_TIMER_TMR_CHANNEL_0 or PDL_ELC_TIMER_TMR_CHANNEL_2	Select the timer to configure.

[data3]

- Timer operation

PDL_ELC_TIMER_COUNT_START or PDL_ELC_TIMER_COUNT_RESTART or PDL_ELC_TIMER_EVENT_COUNTER or PDL_ELC_TIMER_INPUT_CAPTURE or PDL_ELC_TIMER_EVENT_DISABLE	Select the timer operation when triggered.

Operation (data1) = PDL_ELC_PORT_GROUP

[data2]

- Select the port.

PDL_ELC_PORT_B	Select a port whose bits will make up a port group.

[data3]

- Select the bits which define the port group.

b31 – b8	b7-b0
Not used.	Bit mask, where a high bit selects the bit as part of the port group.

Description (4/5)

Operation (data1) = PDL_ELC_PORT_CONTROL

[data2]

- Select a port group

PDL_ELC_PORT_B	Select the port group to control.

[data3]

Set configuration. To set multiple options at the same time, use "|" to separate each value.

- Input Group, edge control.

PDL_ELC_PORT_GROUP_INPUT_RISING_EDGE or PDL_ELC_PORT_GROUP_INPUT_FALLING_EDGE or PDL_ELC_PORT_GROUP_INPUT_ANY_EDGE	If an input port group select the edge that will cause an event.

- Input Group, port buffer control.

PDL_ELC_PORT_GROUP_INPUT_BUFFER_ENABLE or PDL_ELC_PORT_GROUP_INPUT_BUFFER_DISABLE	Enable or disable overwriting of the port data buffer by an input port group.

- Output Group, output control.

PDL_ELC_PORT_GROUP_OUTPUT_0 or PDL_ELC_PORT_GROUP_OUTPUT_1 or PDL_ELC_PORT_GROUP_OUTPUT_TOGGLE or PDL_ELC_PORT_GROUP_OUTPUT_BUFFER or PDL_ELC_PORT_GROUP_OUTPUT_ROTATE	If an output port group configure the output when an event occurs.

Operation (data1) = PDL_ELC_SINGLE_PORT

[data2]

- Single port configuration

PDL_ELC_SINGLE_PORT_0 or PDL_ELC_SINGLE_PORT_1	Select the single port to define / configure.

[data3]

Set configuration. A pin selection and a single port control value must both be specified, use "|" to separate each value.

- Pin selection

PDL_ELC_PIN_PORT_B_0 or PDL_ELC_PIN_PORT_B_1 or PDL_ELC_PIN_PORT_B_2 or PDL_ELC_PIN_PORT_B_3 or PDL_ELC_PIN_PORT_B_4 or PDL_ELC_PIN_PORT_B_5 or PDL_ELC_PIN_PORT_B_6 or PDL_ELC_PIN_PORT_B_7	Select the pin that will be assigned to the single port.

- Single port control

PDL_ELC_PIN_OUTPUT_0 or PDL_ELC_PIN_OUTPUT_1 or PDL_ELC_PIN_OUTPUT_TOGGLE or PDL_ELC_PIN_EVENT_RISING_EDGE or PDL_ELC_PIN_EVENT_FALLING_EDGE or PDL_ELC_PIN_EVENT_ANY_EDGE	Select the single port operation. If the pin is being used as an output pin then the output can be configured. If the pin is being used as an input pin the edge that will cause an event can be configured.

Description (5/5)

Operation (data1) = PDL_ELC_SOFTWARE_EVENT

[data2] Not used. Specify PDL_NO_DATA
[data3] Not used. Specify PDL_NO_DATA

Operation (data1) = PDL_ELC_TRIGGER

[data2]

- DTC Trigger setup. If multiple selections are required, use "|" to separate each selection.

PDL_ELC_INTERRUPT_1_TRIGGER_DTC_ENABLE or PDL_ELC_INTERRUPT_1_TRIGGER_DTC_DISABLE	Enable or disable a DTC trigger from an ELC Interrupt

[data3] Not used. Specify PDL_NO_DATA

Return value

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

Category

Event Link Controller

Reference

R_ELC_Create, R_TMR_ControlChannel

Remarks

- Each time this function is called a particular control operation is performed.
  Call this function as many times as is required to complete the necessary configuration.
- If using an event to start the 8-Bit Timer then to stop it use function R_TMR_ControlChannel with PDL_TMR_ELC_COUNT_STOP selected.
- If another peripheral or port will be used to generate an event or to be triggered by an event then that peripheral or port must be configured accordingly before calling this function.
- Only the following events can be selected to trigger an ELC Interrupt:
  PDL_ELC_LINK_EVENT_INPUT_PORT_GROUP_B or
  PDL_ELC_LINK_EVENT_SINGLE_INPUT_PORT_0 or
  PDL_ELC_LINK_EVENT_SINGLE_INPUT_PORT_1 or
  PDL_ELC_LINK_EVENT_SOFTWARE_EVENT
- An ELC interrupt can trigger the DTC or the DMAC but not both at the same time.
  If triggering the DTC use PDL_ELC_TRIGGER to enable the DTC trigger.
- Some peripheral channels and port pins are not available on some device packages. Please check the hardware manual.
- The event signals selected by PDL_ELC_LINK_EVENT_SCI5_RECEIVE_DATA_FULL and PDL_ELC_LINK_EVENT_SCI5_TRANSMIT_DATA_EMPTY depend on the SCI5 operation mode. Please refer to section 27.12 in the RX220 hardware manual.
- Event PDL_ELC_LINK_EVENT_SCI5_RECEIVE_DATA_FULL cannot be used if SCI5 is configured for IIC Mode. This function will return false if this condition is detected.
- Event PDL_ELC_LINK_EVENT_SPI_TRANSMIT_END cannot be used if RSPI is in slave mode of clock synchronous operation. This function will return false if this condition is detected.
- If a timer event link is no longer required, use PDL_ELC_TIMER_EVENT_DISABLE to disable the event.
- During software standby mode the voltage detection event triggers can be generated but they will not cause a trigger until software standby mode is exited.
- To enable the voltage detection event triggers, enable the LVD first then enable the LVD event link function at the ELC. To disable this function, disable the LVD event link function at the ELC first then disable the LVD.
- For 64-pin package, if port PC0 and PC1 are selected in port switching register A, input to ports PB6 and PB7 and the output port event function of the ELC cannot be used. For 48-pin package, if port PC0 to PC3 are selected in port switching register B, input to ports PB0, PB1, PB3 and PB5 and the output port event function of the ELC cannot be used.
- Event PDL_ELC_LINK_EVENT_SPI_ERROR cannot be used if multi-master configuration, SPI operation, and master mode are selected for the RSPI.

Program example

```
#include "r_pdl_elc.h"

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

void func(void)
{
    /*Create link between event 'TMR Channel 0' and
        module 'SinglePort 0'.*/
    R_ELC_Control(
        PDL_ELC_CREATE_LINK,
        PDL_ELC_LINK_MODULE_SINGLE_PORT_0,
        PDL_ELC_LINK_EVENT_TMR_CHANNEL_0_COMPARE_MATCH_A0
    );

    /*Configure 'SinglePort 0' as PB_0. Toggle output on event. */
    R_ELC_Control(
        PDL_ELC_SINGLE_PORT,
        PDL_ELC_SINGLE_PORT_0,
        PDL_ELC_PIN_PORT_B_0 | PDL_ELC_PIN_OUTPUT_TOGGLE
    );

    /*Enable all Links.*/
    R_ELC_Control(
        PDL_ELC_ENABLE,
        PDL_NO_DATA,
        PDL_NO_DATA
    );
}
```

4.2.14. Multi-Function Timer Pulse Unit

1) R_MTU2_Set

Synopsis

Configure the Multi-function Timer Pulse Unit.

Prototype

```
bool R_MTU2_Set(
    uint8_t data1,      // Channel selection
    uint32_t data2      // Configuration for a channel
);
```

Description (1/2)

Set up the global MTU options.

[data1]
The channel number n (where n = 0 to 5).

[data2]
Pin configuration for the channel. Use "|" to separate each selection.

- Valid when n= 0

PDL_MTU2_PIN_0A_P34 or PDL_MTU2_PIN_0A_PB3	Select the P34 or PB3 pin for MTIOC0A.
PDL_MTU2_PIN_0B_P13 or PDL_MTU2_PIN_0B_P15 or PDL_MTU2_PIN_0B_PA1	Select the P13, P15 or PA1 pin for MTIOC0B.
PDL_MTU2_PIN_0C_P32 or PDL_MTU2_PIN_0C_PB1	Select the P32 or PB1 pin for MTIOC0C.
PDL_MTU2_PIN_0D_P33 or PDL_MTU2_PIN_0D_PA3	Select the P33 or PA3 pin for MTIOC0D.

- Valid when n=1

PDL_MTU2_PIN_1A_P20 or PDL_MTU2_PIN_1A_PE4	Select the P20 or PE4 pin for MTIOC1A.
PDL_MTU2_PIN_1B_P21 or PDL_MTU2_PIN_1B_PB5	Select the P21 or PB5 pin for MTIOC1B.

- Valid when n=2

PDL_MTU2_PIN_2A_P26 or PDL_MTU2_PIN_2A_PB5	Select the P26 or PB5 pin for MTIOC2A.
PDL_MTU2_PIN_2B_P27 or PDL_MTU2_PIN_2B_PE5	Select the P27 or PE5 pin for MTIOC2B.

- Valid when n=3

PDL_MTU2_PIN_3A_P14 or PDL_MTU2_PIN_3A_P17 or PDL_MTU2_PIN_3A_PC1 or PDL_MTU2_PIN_3A_PC7 or PDL_MTU2_PIN_3A_PJ1	Select the P14, P17, PC1, PC7 or PJ1 pin for MTIOC3A.
PDL_MTU2_PIN_3B_P17 or PDL_MTU2_PIN_3B_P22 or PDL_MTU2_PIN_3B_PB7 or PDL_MTU2_PIN_3B_PC5	Select the P17, P22, PB7 or PC5 pin for MTIOC3B.
PDL_MTU2_PIN_3C_P16 or PDL_MTU2_PIN_3C_PC0 or PDL_MTU2_PIN_3C_PC6 or PDL_MTU2_PIN_3C_PJ3	Select the P16, PC0, PC6 or PJ3 pin for MTIOC3C.
PDL_MTU2_PIN_3D_P16 or PDL_MTU2_PIN_3D_P23 or PDL_MTU2_PIN_3D_PB6 or PDL_MTU2_PIN_3D_PC4	Select the P16, P23, PB6 or PC4 pin for MTIOC3D.

Description (2/2)

- Valid when n=4

PDL_MTU2_PIN_4A_P24 or PDL_MTU2_PIN_4A_PA0 or PDL_MTU2_PIN_4A_PB3 or PDL_MTU2_PIN_4A_PE2	Select the P24, PA0, PB3 or PE2 pin for MTIOC4A.
PDL_MTU2_PIN_4B_P30 or PDL_MTU2_PIN_4B_P54 or PDL_MTU2_PIN_4B_PC2 or PDL_MTU2_PIN_4B_PD1 or PDL_MTU2_PIN_4B_PE3	Select the P30, P54, PC2, PD1 or PE3 pin for MTIOC4B.
PDL_MTU2_PIN_4C_P25 or PDL_MTU2_PIN_4C_PB1 or PDL_MTU2_PIN_4C_PE1 or PDL_MTU2_PIN_4C_PE5	Select the P25, PB1, PE1 or PE5 pin for MTIOC4C.
PDL_MTU2_PIN_4D_P31 or PDL_MTU2_PIN_4D_P55 or PDL_MTU2_PIN_4D_PC3 or PDL_MTU2_PIN_4D_PD2 or PDL_MTU2_PIN_4D_PE4	Select the P31, P55, PC3, PD2 or PE4 pin for MTIOC4D.

- Valid when n=5

PDL_MTU2_PIN_5U_PA4 or PDL_MTU2_PIN_5U_PD7	Select the PA4 or PD7 pin for MTIOC5U.
PDL_MTU2_PIN_5V_PA6 or PDL_MTU2_PIN_5V_PD6	Select the PA6 or PD6 pin for MTIOC5V.
PDL_MTU2_PIN_5W_PB0 or PDL_MTU2_PIN_5W_PD5	Select the PB0 or PD5 pin for MTIOC5W.

- Valid when n = 0, 1, 2, 3 or 4

PDL_MTU2_PIN_CLKA_P14 or PDL_MTU2_PIN_CLKA_P24 or PDL_MTU2_PIN_CLKA_PA4 or PDL_MTU2_PIN_CLKA_PC6	Select the P14, P24, PA4 or PC6 pin for MTCLKA.
PDL_MTU2_PIN_CLKB_P15 or PDL_MTU2_PIN_CLKB_P25 or PDL_MTU2_PIN_CLKB_PA6 or PDL_MTU2_PIN_CLKB_PC7	Select the P15, P25, PA6 or PC7 pin for MTCLKB.

- Valid when n = 0 or 2

PDL_MTU2_PIN_CLKC_P22 or PDL_MTU2_PIN_CLKC_PA1 or PDL_MTU2_PIN_CLKC_PC4	Select the P22, PA1 or PC4 pin for MTCLKC.
PDL_MTU2_PIN_CLKD_P23 or PDL_MTU2_PIN_CLKD_PA3 or PDL_MTU2_PIN_CLKD_PC5	Select the P23, PA3 or PC5 pin for MTCLKD. When n = 2, required in Phase Counting Mode only,

Return value

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

Category

Multi-function Timer Pulse Unit

Reference

R_MTU2_Create

Remarks

- Before calling R_MTU2_Create, call this function to configure the relevant pins.
- Make sure no more than one peripheral function is assigned to a single pin.
- Make sure the configuration of MTCLK pins is consistent for all the channels.
- Device packages with 64 or fewer pins do not have all of the pin options.

Program example

```
#include "r_pdl_mtu2.h"

void func(void)
{
    /* Configure the MTU pins */
    R_MTU2_Set(
        0,
        PDL_MTU2_PIN_0A_P34
    );
}
```

2) R_MTU2_Create

Synopsis

Configure an MTU channel.

Prototype

```
bool R_MTU2_Create(
    uint8_t data1,                      // Channel selection
    R_MTU2_Create_structure * data2     // A pointer to the structure
);
```

R_MTU2_Create_structure members:

```
uint32_t channel_mode                   // Configuration selection
uint32_t counter_operation              // Configuration selection
uint32_t ADC_trigger_operation          // Configuration selection
uint16_t buffer_operation               // Configuration selection
uint32_t TGR_A_B_operation              // Configuration selection
uint32_t TGR_C_D_operation              // Configuration selection
uint32_t TGR_U_V_W_operation            // Configuration selection
uint16_t noise_filter_operation         // Configuration selection
uint16_t TCNT_TCNTU_value               // Register value
uint16_t TGRA_TCNTV_value               // Register value
uint16_t TGRB_TCNTW_value               // Register value
uint16_t TGRC_TGRU_value                // Register value
uint16_t TGRD_TGRV_value                // Register value
uint16_t TGRE_TGRW_value                // Register value
uint16_t TGRF_TADCORA_value             // Register value
uint16_t TADCORB_value                  // Register value
uint16_t TADCOBRA_value                 // Register value
uint16_t TADCOBRB_value                 // Register value
void * func1                            // Callback function
void * func2                            // Callback function
void * func3                            // Callback function
void * func4                            // Callback function
uint8_t interrupt_priority_1            // Interrupt priority level
void * func5                            // Callback function
void * func6                            // Callback function
void * func7                            // Callback function
void * func8                            // Callback function
uint8_t interrupt_priority_2            // Interrupt priority level
```

Description (1/9)

Set up a 16-bit MTU2 channel.

[data1]
The channel number n (where n = 0 to 5).

[channel_mode]
Configure the channel mode.
If multiple selections are required, use “|” to separate each selection.
The default settings are shown in **bold**.

- Operation mode. Valid for n = 0 to 4, unless stated otherwise.

PDL_MTU2_MODE_NORMAL or	Normal operation.
PDL_MTU2_MODE_PWM1 or	Pulse Width Modulation (PWM) mode 1.
PDL_MTU2_MODE_PWM2 or	Pulse Width Modulation (PWM) mode 2. Valid for n = 0, 1, and 2.
PDL_MTU2_MODE_PHASE1 or PDL_MTU2_MODE_PHASE2 or PDL_MTU2_MODE_PHASE3 or PDL_MTU2_MODE_PHASE4 or	Phase counting mode 1, 2, 3 or 4. Valid for n = 1 and 2.
PDL_MTU2_MODE_PWM_RS or	Reset-synchronised PWM mode. Valid for n = 3.
PDL_MTU2_MODE_PWM_COMP1 or PDL_MTU2_MODE_PWM_COMP2 or PDL_MTU2_MODE_PWM_COMP3	Complementary PWM mode 1, 2 or 3. Valid for n = 3. Select Normal operation when configuring channel 4.

Description (2/9)

- Synchronous mode. Valid for n = 0 to 4.

PDL_MTU2_SYNC_DISABLE or PDL_MTU2_SYNC_ENABLE	Disable or enable synchronous pre-setting / clearing.

- DMAC / DTC event trigger control. Valid for n = 0 to 4 unless stated otherwise.

PDL_MTU2_TGRA_DMAC_DTC_TRIGGER_DISABLE or PDL_MTU2_TGRA_DMAC_TRIGGER_ENABLE or PDL_MTU2_TGRA_DTC_TRIGGER_ENABLE	TGRA compare match or input capture.
PDL_MTU2_TGRB_DTC_TRIGGER_DISABLE or PDL_MTU2_TGRB_DTC_TRIGGER_ENABLE	TGRB compare match or input capture.
PDL_MTU2_TGRC_DTC_TRIGGER_DISABLE or PDL_MTU2_TGRC_DTC_TRIGGER_ENABLE	TGRC compare match or input capture. Valid for n = 0, 3 and 4.
PDL_MTU2_TGRD_DTC_TRIGGER_DISABLE or PDL_MTU2_TGRD_DTC_TRIGGER_ENABLE	TGRD compare match or input capture. Valid for n = 0, 3 and 4.
PDL_MTU2_TCIV_DTC_TRIGGER_DISABLE or PDL_MTU2_TCIV_DTC_TRIGGER_ENABLE	Counter overflow or underflow. Valid for n = 4.

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

PDL_MTU2_TGRU_DTC_TRIGGER_DISABLE or PDL_MTU2_TGRU_DTC_TRIGGER_ENABLE	TGRU compare match or input capture.
PDL_MTU2_TGRV_DTC_TRIGGER_DISABLE or PDL_MTU2_TGRV_DTC_TRIGGER_ENABLE	TGRV compare match or input capture.
PDL_MTU2_TGRW_DTC_TRIGGER_DISABLE or PDL_MTU2_TGRW_DTC_TRIGGER_ENABLE	TGRW compare match or input capture.

[counter_operation]
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 = 0 to 4 unless stated otherwise.
  Not effective for n = 1 and 2 in Phase Counting Mode.

PDL_MTU2_CLK_PCLK_DIV_1 or PDL_MTU2_CLK_PCLK_DIV_4 or PDL_MTU2_CLK_PCLK_DIV_16 or PDL_MTU2_CLK_PCLK_DIV_64 or	The internal clock signal PCLKB ÷ 1, 4, 16 or 64.
PDL_MTU2_CLK_PCLK_DIV_256 or	PCLKB ÷ 256. Valid for n = 1, 3 and 4.
PDL_MTU2_CLK_PCLK_DIV_1024 or	PCLKB ÷ 1024. Valid for n = 2, 3 and 4.
PDL_MTU2_CLK_MTCLKA or	MTCLKA pin input. Valid for n = 0 to 4.
PDL_MTU2_CLK_MTCLKB or	MTCLKB pin input. Valid for n = 0 to 4.
PDL_MTU2_CLK_MTCLKC or	MTCLKC pin input. Valid for n = 0 or 2.
PDL_MTU2_CLK_MTCLKD or	MTCLKD pin input. Valid for n = 0.
PDL_MTU2_CLK_CASCADE	The overflow / underflow signal from channel (n+1). Valid for n = 1.

- TCNT counter clock edge selection. Valid for n = 0 to 4.Not effective for n = 1 and 2 in Phase Counting Mode.

PDL_MTU2_CLK_RISING or PDL_MTU2_CLK_FALLING or PDL_MTU2_CLK_BOTH	The TCNT counter clock signal shall be counted on rising, falling or both edges.

- TCNT counter clearing. Valid for n = 0 to 4 unless stated otherwise.

PDL_MTU2_CLEAR_DISABLE or	Clearing is disabled.
PDL_MTU2_CLEAR_TGRA or	Cleared by TGRA compare match or input capture.
PDL_MTU2_CLEAR_TGRB or	Cleared by TGRB compare match or input capture.
PDL_MTU2_CLEAR_SYNC or	Cleared by counter clearing on another channel configured for synchronous operation.
PDL_MTU2_CLEAR_TGRC or	Cleared by TGRC compare match or input capture. Valid for n = 0, 3 and 4.
PDL_MTU2_CLEAR_TGRD	Cleared by TGRD compare match or input capture. Valid for n = 0, 3 and 4.

Description (3/9)

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

PDL_MTU2_CLKU_PCLK_DIV_1 or PDL_MTU2_CLKU_PCLK_DIV_4 or PDL_MTU2_CLKU_PCLK_DIV_16 or PDL_MTU2_CLKU_PCLK_DIV_64	Counter TCNTU is supplied by the internal clock signal PCLKB ÷ 1, 4, 16 or 64.
PDL_MTU2_CLKV_PCLK_DIV_1 or PDL_MTU2_CLKV_PCLK_DIV_4 or PDL_MTU2_CLKV_PCLK_DIV_16 or PDL_MTU2_CLKV_PCLK_DIV_64	Counter TCNTV is supplied by the internal clock signal PCLKB ÷ 1, 4, 16 or 64.
PDL_MTU2_CLKW_PCLK_DIV_1 or PDL_MTU2_CLKW_PCLK_DIV_4 or PDL_MTU2_CLKW_PCLK_DIV_16 or PDL_MTU2_CLKW_PCLK_DIV_64	Counter TCNTW is supplied by the internal clock signal PCLKB ÷ 1, 4, 16 or 64.

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

PDL_MTU2_CLEAR_TGRU_DISABLE or PDL_MTU2_CLEAR_TGRU_ENABLE	Disable or enable clearing of TCNTU by TGRU compare match or input capture.
PDL_MTU2_CLEAR_TGRV_DISABLE or PDL_MTU2_CLEAR_TGRV_ENABLE	Disable or enable clearing of TCNTV by TGRV compare match or input capture.
PDL_MTU2_CLEAR_TGRW_DISABLE or PDL_MTU2_CLEAR_TGRW_ENABLE	Disable or enable clearing of TCNTW by TGRW compare match or input capture.

[ADC_trigger_operation]
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 = 0 to 4 unless stated otherwise.

PDL_MTU2_ADC_TRIG_TGRA_DISABLE or PDL_MTU2_ADC_TRIG_TGRA_ENABLE	Disable or enable ADC start requests on a TGRA compare match or input capture.
PDL_MTU2_ADC_TRIG_TROUGH_DISABLE or PDL_MTU2_ADC_TRIG_TROUGH_ENABLE	Disable or enable ADC start requests on a TCNT underflow. Valid for n = 4 in complementary PWM mode.

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

PDL_MTU2_ADC_TRIG_A_TROUGH_INT_SKIP_DISABLE or PDL_MTU2_ADC_TRIG_A_TROUGH_INT_SKIP_ENABLE	Disable or link interrupt skipping to ADC trigger TRGnAN on a TCNT underflow.
PDL_MTU2_ADC_TRIG_B_TROUGH_INT_SKIP_DISABLE or PDL_MTU2_ADC_TRIG_B_TROUGH_INT_SKIP_ENABLE	Disable or link interrupt skipping to ADC trigger TRGnBN on a TCNT underflow.
PDL_MTU2_ADC_TRIG_A_CREST_INT_SKIP_DISABLE or PDL_MTU2_ADC_TRIG_A_CREST_INT_SKIP_ENABLE	Disable or link interrupt skipping to ADC trigger TRGnAN on a TGRA compare match.
PDL_MTU2_ADC_TRIG_B_CREST_INT_SKIP_DISABLE or PDL_MTU2_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 in complementary PWM mode unless stated otherwise.

PDL_MTU2_ADC_TRIG_A_DOWN_DISABLE or PDL_MTU2_ADC_TRIG_A_DOWN_ENABLE	Disable or enable ADC trigger TRGnAN requests during down-count operation.
PDL_MTU2_ADC_TRIG_B_DOWN_DISABLE or PDL_MTU2_ADC_TRIG_B_DOWN_ENABLE	Disable or enable ADC trigger TRGnBN requests during down-count operation.
PDL_MTU2_ADC_TRIG_A_UP_DISABLE or PDL_MTU2_ADC_TRIG_A_UP_ENABLE	Disable or enable ADC trigger TRGnAN requests during up-count operation. This option can be selected in other modes.
PDL_MTU2_ADC_TRIG_B_UP_DISABLE or PDL_MTU2_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 (4/9)

[buffer_operation]
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.

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

PDL_MTU2_CSB_DISABLE or PDL_MTU2_CSB_CREST or PDL_MTU2_CSB_TROUGH or PDL_MTU2_CSB_BOTH	Select no transfer, transfer on crest detection, transfer on trough detection or transfer on crest and trough detection.

PDL_MTU2_CSB_TROUGH and PDL_MTU2_CSB_BOTH are available only in complementary PWM mode.

- Buffer operation

PDL_MTU2_BUFFER_AC_DISABLE or PDL_MTU2_BUFFER_AC_ENABLE	Disable or enable buffer operation for registers TGRA and TGRC. Valid for n = 0 ,3 and 4.
PDL_MTU2_BUFFER_BD_DISABLE or PDL_MTU2_BUFFER_BD_ENABLE	Disable or enable buffer operation for registers TGRB and TGRD. Valid for n = 0 ,3 and 4.
PDL_MTU2_BUFFER_EF_DISABLE or PDL_MTU2_BUFFER_EF_ENABLE	Disable or enable buffer operation for registers TGRE and TGRF. Valid for n = 0.

- Buffer data transfer

PDL_MTU2_BUFFER_AC_CM_A or PDL_MTU2_BUFFER_AC_TCNT_CLR	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 and 4.
PDL_MTU2_BUFFER_BD_CM_B or PDL_MTU2_BUFFER_BD_TCNT_CLR	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 and 4.
PDL_MTU2_BUFFER_EF_CM_E or PDL_MTU2_BUFFER_EF_TCNT_CLR	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.

Transfer on TCNT clear is available only in PWM mode 1 or 2.

Description (5/9)

[TGR_A_B_operation]
Configure the operation for general registers TGRA and TGRB. Valid for n = 0 to 4.
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

PDL_MTU2_A_OC_DISABLED or PDL_MTU2_A_OC_LOW or PDL_MTU2_A_OC_LOW_CM_HIGH or PDL_MTU2_A_OC_LOW_CM_INV or PDL_MTU2_A_OC_HIGH_CM_LOW or PDL_MTU2_A_OC_HIGH or PDL_MTU2_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_MTU2_A_IC_RISING_EDGE or PDL_MTU2_A_IC_FALLING_EDGE or PDL_MTU2_A_IC_BOTH_EDGES or	Input capture at MTIOCnA rising edge. Input capture at MTIOCnA falling edge. Input capture at MTIOCnA both edges.
PDL_MTU2_A_IC_COUNT or	Input capture at channel (n+1) up-count or down-count. Valid only for n = 0.
PDL_MTU2_A_IC_CM_IC	Input capture at channel (n-1) TGRC compare match or input capture. Valid only for n = 1.

- Input capture / output compare control for register TGRB.

PDL_MTU2_B_OC_DISABLED or PDL_MTU2_B_OC_LOW or PDL_MTU2_B_OC_LOW_CM_HIGH or PDL_MTU2_B_OC_LOW_CM_INV or PDL_MTU2_B_OC_HIGH_CM_LOW or PDL_MTU2_B_OC_HIGH or PDL_MTU2_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_MTU2_B_IC_RISING_EDGE or PDL_MTU2_B_IC_FALLING_EDGE or PDL_MTU2_B_IC_BOTH_EDGES or	Input capture at MTIOCnB rising edge. Input capture at MTIOCnB falling edge. Input capture at MTIOCnB both edges.
PDL_MTU2_B_IC_COUNT or	Input capture at channel (n+1) up-count or down-count. Valid only for n = 0.
PDL_MTU2_B_IC_CM_IC	Input capture at channel (n-1) TGRC compare match or input capture. Valid only for n = 1.

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

PDL_MTU2_CASCADE_AL_IC_EXC_H or PDL_MTU2_CASCADE_AL_IC_INC_H	Exclude or include pin MTIOCnA in the TGRA input capture conditions for channel (n+1).
PDL_MTU2_CASCADE_BL_IC_EXC_H or PDL_MTU2_CASCADE_BL_IC_INC_H	Exclude or include pin MTIOCnB in the TGRB input capture conditions for channel (n+1).
PDL_MTU2_CASCADE_AH_IC_EXC_L or PDL_MTU2_CASCADE_AH_IC_INC_L	Exclude or include pin MTIOC(n+1)A in the TGRA input capture conditions for channel n.
PDL_MTU2_CASCADE_BH_IC_EXC_L or PDL_MTU2_CASCADE_BH_IC_INC_L	Exclude or include pin MTIOC(n+1)B in the TGRB input capture conditions for channel n.

Description (6/9)

[TGR_C_D_operation]
Configure the operation for general registers TGRC and TGRD. Valid for n = 0, 3 and 4.
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.

PDL_MTU2_C_OC_DISABLED or PDL_MTU2_C_OC_LOW or PDL_MTU2_C_OC_LOW_CM_HIGH or PDL_MTU2_C_OC_LOW_CM_INV or PDL_MTU2_C_OC_HIGH_CM_LOW or PDL_MTU2_C_OC_HIGH or PDL_MTU2_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_MTU2_C_IC_RISING_EDGE or PDL_MTU2_C_IC_FALLING_EDGE or PDL_MTU2_C_IC_BOTH_EDGES or	Input capture at MTIOCnC rising edge. Input capture at MTIOCnC falling edge. Input capture at MTIOCnC both edges.
PDL_MTU2_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.

PDL_MTU2_D_OC_DISABLED or PDL_MTU2_D_OC_LOW or PDL_MTU2_D_OC_LOW_CM_HIGH or PDL_MTU2_D_OC_LOW_CM_INV or PDL_MTU2_D_OC_HIGH_CM_LOW or PDL_MTU2_D_OC_HIGH or PDL_MTU2_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_MTU2_D_IC_RISING_EDGE or PDL_MTU2_D_IC_FALLING_EDGE or PDL_MTU2_D_IC_BOTH_EDGES or	Input capture at MTIOCnD rising edge. Input capture at MTIOCnD falling edge. Input capture at MTIOCnD both edges.
PDL_MTU2_D_IC_COUNT	Input capture at channel (n+1) up-count or down-count. Valid only for n = 0.

Description (7/9)

[TGR_U_V_W_operation]
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.

PDL_MTU2_U_CM or	Compare match.
PDL_MTU2_U_IC_RISING_EDGE or PDL_MTU2_U_IC_FALLING_EDGE or PDL_MTU2_U_IC_BOTH_EDGES or	Input capture at MTICnU rising edge. Input capture at MTICnU falling edge. Input capture at MTICnU both edges.
PDL_MTU2_U_IC_PWM_LOW_TROUGH or PDL_MTU2_U_IC_PWM_LOW_CREST or PDL_MTU2_U_IC_PWM_LOW_BOTH or	Input capture at trough, crest or both for low pulse width measurement.
PDL_MTU2_U_IC_PWM_HIGH_TROUGH or PDL_MTU2_U_IC_PWM_HIGH_CREST or PDL_MTU2_U_IC_PWM_HIGH_BOTH	Input capture at trough, crest or both for high pulse width measurement.

- Input capture / compare match control for register TGRV.

PDL_MTU2_V_CM or	Compare match.
PDL_MTU2_V_IC_RISING_EDGE or PDL_MTU2_V_IC_FALLING_EDGE or PDL_MTU2_V_IC_BOTH_EDGES or	Input capture at MTICnV rising edge. Input capture at MTICnV falling edge. Input capture at MTICnV both edges.
PDL_MTU2_V_IC_PWM_LOW_TROUGH or PDL_MTU2_V_IC_PWM_LOW_CREST or PDL_MTU2_V_IC_PWM_LOW_BOTH or	Input capture at trough, crest or both for low pulse width measurement.
PDL_MTU2_V_IC_PWM_HIGH_TROUGH or PDL_MTU2_V_IC_PWM_HIGH_CREST or PDL_MTU2_V_IC_PWM_HIGH_BOTH	Input capture at trough, crest or both for high pulse width measurement.

- Input capture / compare match control for register TGRW.

PDL_MTU2_W_CM or	Compare match.
PDL_MTU2_W_IC_RISING_EDGE or PDL_MTU2_W_IC_FALLING_EDGE or PDL_MTU2_W_IC_BOTH_EDGES or	Input capture at MTICnW rising edge. Input capture at MTICnW falling edge. Input capture at MTICnW both edges.
PDL_MTU2_W_IC_PWM_LOW_TROUGH or PDL_MTU2_W_IC_PWM_LOW_CREST or PDL_MTU2_W_IC_PWM_LOW_BOTH or	Input capture at trough, crest or both for low pulse width measurement.
PDL_MTU2_W_IC_PWM_HIGH_TROUGH or PDL_MTU2_W_IC_PWM_HIGH_CREST or PDL_MTU2_W_IC_PWM_HIGH_BOTH	Input capture at trough, crest or both for high pulse width measurement.

Description (8/9)

[noise_filter_operation]
Noise filter control for register NFCRn (n = 0 to 5)
The default settings are shown in **bold**. Specify PDL_NO_DATA to use the defaults.

- Noise filter control for register NFCRn

PDL_MTU2_NF_A_U_DISABLE or PDL_MTU2_NF_A_U_ENABLE	Enable or disable noise filter for MTIOCnA (n = 0 to 4) or TIOC5U (n = 5).
PDL_MTU2_NF_B_V_DISABLE or PDL_MTU2_NF_B_V_ENABLE	Enable or disable noise filter for MTIOCnB (n = 0 to 4) or TIOC5V (n = 5).
PDL_MTU2_NF_C_W_DISABLE or PDL_MTU2_NF_C_W_ENABLE	Enable or disable noise filter for MTIOCnC (n = 0, 3 or 4) or TIOC5W (n = 5). Not valid for n=1 or 2.
PDL_MTU2_NF_D_DISABLE or PDL_MTU2_NF_D_ENABLE	Enable or disable noise filter for MTIOCnD (n = 0, 3 or 4). Not valid for n = 1, 2 or 5.

- Noise filter clock select for register NFCRn

PDL_MTU2_NF_PCLK_DIV_1 or PDL_MTU2_NF_PCLK_DIV_8 or PDL_MTU2_NF_PCLK_DIV_32 or PDL_MTU2_NF_PCLK_DIV_SRC	Set the clock of the noise filter as PCLKB ÷ 1, 8, 32 or the count source.

[TCNT_TCNTU_value]
For n = 0 to 4: The timer counter TCNT value.
For n = 5: The timer counter TCNTU value.

[TGRA_TCNTV_value]
For n = 0 to 4: The register TGRA value.
For n = 5: The timer counter TCNTV value.

[TGRB_TCNTW_value]
For n = 0 to 4: The register TGRB value.
For n = 5: The timer counter TCNTW value.

[TGRC_TGRU_value]
For n = 0, 3 or 4: The register TGRC value.
For n = 5: The register TGRU value. If the corresponding channel is stopped, make sure the value is not TCNTU + 1.
Ignored for n = 1 or 2.

[TGRD_TGRV_value]
For n = 0, 3 or 4: The register TGRD value.
For n = 5: The register TGRV value. If the corresponding channel is stopped, make sure the value is not TCNTV + 1.
Ignored for n = 1 or 2.

[TGRE_TGRW_value]
For n = 0: The register TGRE value.
For n = 5: The register TGRW value. If the corresponding channel is stopped, make sure the value is not TCNTW + 1.
Ignored for n = 1, 2, 3 or 4.

[TGRF_TADCORA_value]
For n = 0: The register TGRF value.
For n = 4: The register TADCORA value.
Ignored for n = 1, 2, 3 or 5.

[TADCORB_value]
The register TADCORB value (ignored for n ≠ 4).

[TADCOBRA_value]
The register TADCOBRA value (ignored for n ≠ 4).

[TADCOBRB_value]
The register TADCOBRB value (ignored for n ≠ 4).

Description (9/9)

[func1]
For n = 0 to 4: 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 = 0 to 4: 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 or 4: 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 or 4: The function to be called when a TGRD event occurs.
Specify PDL_NO_FUNC if not required.

[interrupt_priority_1]
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: 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 or 2: The function to be called when an underflow occurs.
Specify PDL_NO_FUNC if not required.

[interrupt_priority_2]
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).

Return value

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

Category

Multi-function Timer Pulse Unit

Reference

R_MTU2_Set, R_MTU2_ControlChannel, R_MTU2_ControlUnit

Remarks

- 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.
- The alternative pins are assigned using function R_MTU2_Set.
- Either R_MTU2_ControlChannel or R_MTU2_ControlUnit must be used to start the timers.
- If a callback function is specified, this function will enable the relevant CPU interrupt. Please see the notes on callback function usage 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 the channel is configured for phase counting mode, the counter clock source setting is ignored.
- If buffer operation is selected for registers TGRA and TGRC, input capture / output compare is not valid for register TGRC.
- If buffer operation is selected for registers TGRB and TGRD, input capture / output compare is not valid for register TGRD.
- If synchronous mode is required, at least two channels must be enabled for synchronous operation.
- A companion function, R_MTU2_Create_load_defaults, can be used to load the default values into the structure.
- If the channel operation mode will be changed, ensure that the timer is stopped (use R_MTU2_ControlChannel or R_MTU2_ControlUnit).
- If the noise filter is enabled, wait for 2 cycles of the selected noise filter clock before starting the timer (use R_MTU2_ControlChannel or R_MTU2_ControlUnit).

Program example

```
/* RPDL definitions */
#include "r_pdl_mtu2.h"

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

void func(void)
{
    /* Allocate a copy of the structure for the selected channel */
    R_MTU2_Create_structure ch4_parameters;

    /* Load the defaults */
    R_MTU2_Create_load_defaults(&ch4_parameters);

    /* Set the non-default options for channel 4 */
    ch4_parameters.channel_mode= PDL_MTU2_MODE_NORMAL | \
            PDL_MTU2_SYNC_ENABLE |   PDL_MTU2_TGRA_DTC_TRIGGER_ENABLE;
    ch4_parameters.counter_operation= PDL_MTU2_CLK_PCLK_DIV_4;
    ch4_parameters.buffer_operation= PDL_MTU2_BUFFER_AC_CM_A;
    ch4_parameters.TGR_C_D_operation= PDL_MTU2_C_OC_HIGH_CM_LOW;
    ch4_parameters.TCNT_TCNTU_value= 0;
    ch4_parameters.TGRA_TCNTV_value= 199;
    ch4_parameters.TGRB_TCNTW_value= 99;
    ch4_parameters.TGRC_TGRU_value= 50;
    ch4_parameters.TGRD_TGRV_value= 100;
    ch4_parameters.TGRE_TGRW_value= 0;
    ch4_parameters.TGRF_TADCORA_value= 0;

    R_MTU2_Create(
        4,
       &ch4_parameters
    );
 }
```

3) R_MTU2_Destroy

Synopsis	Disable a Multi-function Timer Pulse Unit.
Prototype	**bool R_MTU2_Destroy(** **uint8_t data** // Unit selection **);**
Description	Shut down a timer pulse unit **[data]** The multi-function timer pulse unit n (where n = 0). Unit 0 comprises channels 0 to 5.
Return value	True if the unit selection is valid; otherwise false.
Category	Multi-function Timer Pulse Unit
Reference	None.
Remarks	• The unit is put into the stop state to reduce power consumption.
Program example	

```
#include "r_pdl_mtu2.h"

void func(void)
{
    /* Shutdown MTU2 channels 0 to 5 */
    R_MTU2_Destroy(
        0
    );
}
```

4) R_MTU2_ControlChannel

Synopsis

Control an MTU channel.

Prototype

```
bool R_MTU2_ControlChannel(
    uint8_t data1,                                  // Channel selection
    R_MTU2_ControlChannel_structure * data2         // A pointer to the structure
);
```

R_MTU2_ControlChannel_structure members:

```
uint8_t control_setting          // Control settings
uint16_t register_selection      // Register selection
uint16_t TCNT_TCNTU_value        // Register value
uint16_t TGRA_TCNTV_value        // Register value
uint16_t TGRB_TCNTW_value        // Register value
uint16_t TGRC_TGRU_value         // Register value
uint16_t TGRD_TGRV_value         // Register value
uint16_t TGRE_TGRW_value         // Register value
uint16_t TGRF_value              // Register value
uint16_t TADCOBRA_value          // Register value
uint16_t TADCOBRB_value          // Register value
```

Description (1/2)

Modify a timer channel's registers.

[data1]
The channel number n (where n = 0 to 5).

[control_setting]
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 = 0 to 4.

PDL_MTU2_STOP	Stop the count operation.
PDL_MTU2_START	Start the count operation.

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

PDL_MTU2_STOP_U	Stop the count operation.
PDL_MTU2_STOP_V	
PDL_MTU2_STOP_W	
PDL_MTU2_START_U	Start the count operation.
PDL_MTU2_START_V	
PDL_MTU2_START_W	

[register_selection]
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 = 0 to 4.

PDL_MTU2_REGISTER_COUNTER	Timer counter register (TCNT).
PDL_MTU2_REGISTER_TGRA	General register A.
PDL_MTU2_REGISTER_TGRB	General register B.
PDL_MTU2_REGISTER_TGRC	General register C. Valid for n = 0, 3 or 4.
PDL_MTU2_REGISTER_TGRD	General register D. Valid for n = 0, 3 or 4.
PDL_MTU2_REGISTER_TGRE	General register E. Valid for n = 0.
PDL_MTU2_REGISTER_TGRF	General register F. Valid for n = 0.
PDL_MTU2_REGISTER_TADCOBRA	ADC start request cycle set buffer A. Valid for n = 4.
PDL_MTU2_REGISTER_TADCOBRB	ADC start request cycle set buffer B. Valid for n = 4.

Description (2/2)

For n = 5.

PDL_MTU2_REGISTER_COUNTER_U	Timer counter U register (TCNTU).
PDL_MTU2_REGISTER_COUNTER_V	Timer counter V register (TCNTV).
PDL_MTU2_REGISTER_COUNTER_W	Timer counter W register (TCNTW).
PDL_MTU2_REGISTER_TGRU	General registerU.
PDL_MTU2_REGISTER_TGRV	General register V.
PDL_MTU2_REGISTER_TGRW	General register W.

[TCNT_TCNTU_value]
For n = 0 to 4: The timer counter TCNT value.
For n = 5: The timer counter TCNTU value.
This will be ignored if the register is not selected.

[TGRA_TCNTV_value]
For n = 0 to 4: The register TGRA value.
For n = 5: The timer counter TCNTV value.
This will be ignored if the register is not selected.

[TGRB_TCNTW_value]
For n = 0 to 4: The register TGRB value.
For n = 5: The timer counter TCNTW value.
This will be ignored if the register is not selected.

[TGRC_TGRU_value]
For n = 0, 3 or 4: The register TGRC value.
For n = 5: The register TGRU value. If the corresponding channel is stopped, make sure the value is not TCNTU + 1.
This will be ignored if the register is not selected.

[TGRD_TGRV_value]
For n = 0, 3 or 4: The register TGRD value.
For n = 5: The register TGRV value. If the corresponding channel is stopped, make sure the value is not TCNTV + 1.
This will be ignored if the register is not selected.

[TGRE_TGRW_value]
For n = 0: The register TGRE value.
For n = 5: The register TGRW value. If the corresponding channel is stopped, make sure the value is not TCNTW + 1.
This will be ignored if the register is not selected.

[TGRF_value]
For n = 0: The general register TGRF value.
This will be ignored if the register is not selected.

[TADCOBRA_value]
For n = 4: ADC start request cycle set buffer A.
This will be ignored if the register is not selected.

[TADCOBRB_value]
For n = 4: 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_MTU2_Create, R_MTU2_ControlUnit

Remarks

- Before calling this function, use R_MTU2_Create to configure the channel operation.
- Either this function or R_MTU2_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.
- If noise filter is enabled, before starting the timer, make sure at least 2 cycles of the selected noise filter clock has elapsed after the timer configuration (use R_MTU2_Create).

Program example

```
/* RPDL definitions */
#include "r_pdl_mtu2.h"

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

void func(void)
{
    /* Allocate a copy of the structure for the selected channel */
    R_MTU2_ControlChannel_structure ch3_parameters;

    /* Set the control options for channel 3 */
    ch3_parameters.control_setting= PDL_MTU2_START;
    ch3_parameters.register_selection= PDL_MTU2_REGISTER_COUNTER |
PDL_MTU2_REGISTER_TGRB;
    ch3_parameters.TCNT_TCNTU_value= 0xFFDD;
    ch3_parameters.TGRB_TCNTW_value= 0x0020;

    /* Modify the operation of channel 3 */
    R_MTU2_ControlChannel(
        3,
        &ch3_parameters
    );
}
```

5) R_MTU2_ControlUnit

Synopsis

Control a Multi-function Timer Pulse Unit.

Prototype

```
bool R_MTU2_ControlUnit(
    uint8_t data1,                          // Unit selection
    R_MTU2_ControlUnit_structure * data2    // A pointer to the structure
);
```

R_MTU2_ControlUnit_structure members:

```
uint16_t simultaneous_control       // Control selection
uint32_t output_control             // Control selection
uint32_t buffer_control             // Control selection
uint16_t brushless_DC_motor_control // Control selection
uint32_t general_control            // Control selection
uint8_t register_selection          // Register selection
uint16_t TDDR_value                 // Register value
uint16_t TCDR_value                 // Register value
uint16_t TCBR_value                 // Register value
```

Description (1/4)

Modify a timer unit's registers.

[data1]
The unit number n (where n = 0).

[simultaneous_control]
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_MTU2_STOP_CH_0	Stop the count operation for the selected channels.
PDL_MTU2_STOP_CH_1	
PDL_MTU2_STOP_CH_2	
PDL_MTU2_STOP_CH_3	
PDL_MTU2_STOP_CH_4	

- Counter start control

PDL_MTU2_START_CH_0	Start the count operation for the selected channels.
PDL_MTU2_START_CH_1	
PDL_MTU2_START_CH_2	
PDL_MTU2_START_CH_3	
PDL_MTU2_START_CH_4	

Description (2/4)

[output_control]
The output control settings to be modified. All settings are optional.
If multiple selections are required, use "|" to separate each selection.

- Output control.
  To apply output control, make sure the operation of the corresponding channel is stopped.

  Select one option for each output.

PDL_MTU2_OUT_P_PHASE_1_ENABLE or PDL_MTU2_OUT_P_PHASE_1_DISABLE	MTIOC3B
PDL_MTU2_OUT_N_PHASE_1_ENABLE or PDL_MTU2_OUT_N_PHASE_1_DISABLE	MTIOC3D
PDL_MTU2_OUT_P_PHASE_2_ENABLE or PDL_MTU2_OUT_P_PHASE_2_DISABLE	MTIOC4A
PDL_MTU2_OUT_N_PHASE_2_ENABLE or PDL_MTU2_OUT_N_PHASE_2_DISABLE	MTIOC4C
PDL_MTU2_OUT_P_PHASE_3_ENABLE or PDL_MTU2_OUT_P_PHASE_3_DISABLE	MTIOC4B.
PDL_MTU2_OUT_N_PHASE_3_ENABLE or PDL_MTU2_OUT_N_PHASE_3_DISABLE	MTIOC4D

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

PDL_MTU2_OUT_P_PHASE_ALL_ENABLE or PDL_MTU2_OUT_P_PHASE_ALL_DISABLE	All P phase outputs.
PDL_MTU2_OUT_N_PHASE_ALL_ENABLE or PDL_MTU2_OUT_N_PHASE_ALL_DISABLE	All N phase outputs.

- Output inversion control (applies only to reset-synchronised or complementary PWM modes).
  Each phase output can be configured for
  a) initial high level, active low level or
  b) initial low level, active high level.
  If dead time is not generated, the options for negative phases will be ignored as their outputs are always the inverse of the positive phases.

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

PDL_MTU2_OUT_P_PHASE_ALL_HIGH_LOW or PDL_MTU2_OUT_P_PHASE_ALL_LOW_HIGH	Positive-phase outputs.
PDL_MTU2_OUT_N_PHASE_ALL_HIGH_LOW or PDL_MTU2_OUT_N_PHASE_ALL_LOW_HIGH	Negative-phase outputs.

  Or independently by selecting one option for each required output.

PDL_MTU2_OUT_P_PHASE_1_HIGH_LOW or PDL_MTU2_OUT_P_PHASE_1_LOW_HIGH	MTIOC3B
PDL_MTU2_OUT_N_PHASE_1_HIGH_LOW or PDL_MTU2_OUT_N_PHASE_1_LOW_HIGH	MTIOC3D
PDL_MTU2_OUT_P_PHASE_2_HIGH_LOW or PDL_MTU2_OUT_P_PHASE_2_LOW_HIGH	MTIOC4A
PDL_MTU2_OUT_N_PHASE_2_HIGH_LOW or PDL_MTU2_OUT_N_PHASE_2_LOW_HIGH	MTIOC4C
PDL_MTU2_OUT_P_PHASE_3_HIGH_LOW or PDL_MTU2_OUT_P_PHASE_3_LOW_HIGH	MTIOC4B
PDL_MTU2_OUT_N_PHASE_3_HIGH_LOW or PDL_MTU2_OUT_N_PHASE_3_LOW_HIGH	MTIOC4D

- Write access control (applies only to reset-synchronised or complementary PWM modes).

PDL_MTU2_OUT_LOCK_ENABLE	Prevent further changes to the phase output control.

- Toggle output control (applies only to reset-synchronised or complementary PWM modes).

PDL_MTU2_OUT_TOGGLE_ENABLE or PDL_MTU2_OUT_TOGGLE_DISABLE	Enable or disable toggle output synchronised with the PWM cycle.

Description (3/4)

[buffer_control]
The buffer control settings to be modified. All settings are optional.
If multiple selections are required, use "|" to separate each selection.

- Output level buffer control (applies only to reset-synchronised or complementary PWM modes). To apply output control, make sure the operation of the corresponding channel is stopped.
  Set the output control to be transferred to the output:

PDL_MTU2_OUT_BUFFER_P_PHASE_1_LOW or PDL_MTU2_OUT_BUFFER_P_PHASE_1_HIGH	MTIOC3B
PDL_MTU2_OUT_BUFFER_N_PHASE_1_LOW or PDL_MTU2_OUT_BUFFER_N_PHASE_1_HIGH	MTIOC3D
PDL_MTU2_OUT_BUFFER_P_PHASE_2_LOW or PDL_MTU2_OUT_BUFFER_P_PHASE_2_HIGH	MTIOC4A
PDL_MTU2_OUT_BUFFER_N_PHASE_2_LOW or PDL_MTU2_OUT_BUFFER_N_PHASE_2_HIGH	MTIOC4C
PDL_MTU2_OUT_BUFFER_P_PHASE_3_LOW or PDL_MTU2_OUT_BUFFER_P_PHASE_3_HIGH	MTIOC4B
PDL_MTU2_OUT_BUFFER_N_PHASE_3_LOW or PDL_MTU2_OUT_BUFFER_N_PHASE_3_HIGH	MTIOC4D

- Set the transfer timing
  In complementary PWM modes:

PDL_MTU2_OUT_BUFFER_TRANSFER_DISABLE or PDL_MTU2_OUT_BUFFER_TRANSFER_CREST or PDL_MTU2_OUT_BUFFER_TRANSFER_TROUGH or PDL_MTU2_OUT_BUFFER_TRANSFER_BOTH	Disable or enable on detection of crest, trough or both

  In Reset-synchronised PWM mode:

PDL_MTU2_OUT_BUFFER_TRANSFER_DISABLE or PDL_MTU2_OUT_BUFFER_TRANSFER_CLEAR	Disable or enable on counter clear.

- Buffer transfer to temporary transfer control. Applicable for complementary PWM modes.

PDL_MTU2_BUFFER_TRANSFER_DISABLE or PDL_MTU2_BUFFER_TRANSFER_ENABLE or PDL_MTU2_BUFFER_TRANSFER_LINK	Disable transfers, enable without linking to interrupt skipping or enable and link to interrupt skipping.

[brushless_DC_motor_control]
Brushless DC motor control settings. All settings are optional.
If multiple selections are required, use "|" to separate each selection.
Applies only to reset-synchronised or complementary PWM modes

- Brushless DC motor waveform control

PDL_MTU2_BDCM_ENABLE or PDL_MTU2_BDCM_DISABLE	Enable or disable brushless DC motor control
PDL_MTU2_BDCM_P_PHASE_ENABLE or PDL_MTU2_BDCM_P_PHASE_DISABLE	Enable or disable PWM outputs on the positive-phase output pins.
PDL_MTU2_BDCM_N_PHASE_ENABLE or PDL_MTU2_BDCM_N_PHASE_DISABLE	Enable or disable PWM outputs on the negative-phase output pins.
PDL_MTU2_BDCM_OPS_FB or	Use input capture signals for output switch control, or
PDL_MTU2_BDCM_OPS_000 or PDL_MTU2_BDCM_OPS_001 or PDL_MTU2_BDCM_OPS_010 or PDL_MTU2_BDCM_OPS_011 or PDL_MTU2_BDCM_OPS_100 or PDL_MTU2_BDCM_OPS_101 or PDL_MTU2_BDCM_OPS_110 or PDL_MTU2_BDCM_OPS_111	Set the outputs according to table 21.39 in the hardware manual.

Description (4/4)

[general_control]
General control settings. All settings are optional.
If multiple selections are required, use "|" to separate each selection.

- Interrupt skipping control

PDL_MTU2_INT_SKIP_TROUGH_DISABLE or PDL_MTU2_INT_SKIP_TROUGH_1 or PDL_MTU2_INT_SKIP_TROUGH_2 or PDL_MTU2_INT_SKIP_TROUGH_3 or PDL_MTU2_INT_SKIP_TROUGH_4 or PDL_MTU2_INT_SKIP_TROUGH_5 or PDL_MTU2_INT_SKIP_TROUGH_6 or PDL_MTU2_INT_SKIP_TROUGH_7	Disable TCNT underflow (TCIV) interrupt skipping, or set the skip count between 1 and 7.
PDL_MTU2_INT_SKIP_CREST_DISABLE or PDL_MTU2_INT_SKIP_CREST_1 or PDL_MTU2_INT_SKIP_CREST_2 or PDL_MTU2_INT_SKIP_CREST_3 or PDL_MTU2_INT_SKIP_CREST_4 or PDL_MTU2_INT_SKIP_CREST_5 or PDL_MTU2_INT_SKIP_CREST_6 or PDL_MTU2_INT_SKIP_CREST_7	Disable TGRA compare match (TGIA) interrupt skipping, or set the skip count between 1 and 7.

- Dead time generation control (applies only to complementary PWM modes).

PDL_MTU2_DEAD_TIME_DISABLE or PDL_MTU2_DEAD_TIME_ENABLE	Disable or enable dead time generation.

- Waveform retention control (applies only to complementary PWM modes).

PDL_MTU2_WAVEFORM_RETAIN_DISABLE or PDL_MTU2_WAVEFORM_RETAIN_ENABLE	Disable or enable waveform output retention.

- Compare match clearing control (applies only to complementary PWM modes 1).

PDL_MTU2_CNT_CLEAR_CM_A_DISABLE or PDL_MTU2_CNT_CLEAR_CM_A_ENABLE	Disable or enable counter clearing on TGRA compare match.

- Reset-synchronised or complementary PWM control

PDL_MTU2_PWM_RS_COMP_ENABLE	Enable reset-synchronised or complementary PWM mode.

- Register protection

PDL_MTU2_ACCESS_DISABLE	Control access to the registers and counters in channels 3 and 4.
PDL_MTU2_ACCESS_ENABLE	

[register_selection]
The unit registers to be modified.
If multiple selections are required, use "|" to separate each selection.

- The registers to be modified. These apply only to complementary PWM mode.

PDL_MTU2_REGISTER_DEAD_TIME	Update the dead time data register (TDDR).
PDL_MTU2_REGISTER_CYCLE_DATA	Update the cycle data register (TCDR).
PDL_MTU2_REGISTER_CYCLE_BUFFER	Update the cycle buffer register (TCBR).

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

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

[TCBR_value]
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_MTU2_ControlChannel

Remarks

- Either this function or R_MTU2_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.
- If the noise filter is enabled, before starting the timer, make sure at least 2 cycles of the selected noise filter clock has elapsed after the timer configuration (use R_MTU2_Create).
- When generating PWM waveforms in complementary PWM mode 1 to complementary PWM mode 3, set the timer cycle data registers (TCDR) and timer dead time data registers (TDDR) to values that satisfy the following condition:
  Timer cycle data register value > Timer dead time data register value × 2 + 2

Program example

```
/* RPDL definitions */
#include "r_pdl_mtu2.h"

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

void func(void)
{
    /* Allocate a copy of the structure for the selected channel */
    R_MTU2_ControlUnit_structure unit0_parameters;

    unit0_parameters.simultaneous_control= PDL_MTU2_START_CH_0 |
                        PDL_MTU2_START_CH_1;
    unit0_parameters.output_control=
PDL_MTU2_OUT_P_PHASE_ALL_HIGH_LOW;
    unit0_parameters.general_control= PDL_MTU2_DEAD_TIME_ENABLE;
    unit0_parameters.register_selection= PDL_MTU2_REGISTER_DEAD_TIME|
                    PDL_MTU2_REGISTER_CYCLE_DATA;
    unit0_parameters.TDDR_value= 0xFFDD;
    unit0_parameters.TCDR_value= 0x0100;

    /* Modify the operation of unit 0 */
    R_MTU2_ControlUnit(
        0,
        &unit0_parameters
    );
}
```

6) R_MTU2_ReadChannel

Synopsis

Read from MTU channel registers.

Prototype

```
bool R_MTU2_ReadChannel(
    uint8_tdata1,       // 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 5).

[data2]
The status flags shall be stored in the format below.
The input capture / compare match flags will be set to1 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
Detection							Count direction
Overflow	Input capture / compare match						
V	F	E	D	C	B	A	0: down 1: up

For n = 1 or 2

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

For n = 3

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

For n = 4

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

For n = 5

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

Description (2/2)

[data3]
For n = 0 to 4: 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.

[data4]
For n = 0 to 4: 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.

[data5]
For n = 0 to 4: 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.

[data6]
For n = 0, 3 or 4: 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.

[data7]
For n = 0, 3 or 4: 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.

[data8]
For n = 0: 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.

[data9]
For n = 0: A pointer to where the TGRF register value shall be stored.
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

None.

Remarks

- If the flags are read, any detection flag that has been set to 1 shall be automatically cleared to 0 by this function.

Program example

```
/* RPDL definitions */
#include "r_pdl_mtu2.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_MTU2_ReadChannel(
        3,
        &Flags,
        PDL_NO_PTR,
        &General_A,
        PDL_NO_PTR,
        PDL_NO_PTR,
        &General_D,
        PDL_NO_PTR,
        PDL_NO_PTR
    );
}
```

7) R_MTU2_ReadUnit

Synopsis

Read from MTU registers.

Prototype

```
bool R_MTU2_ReadUnit(
    uint8_tdata1,       // Unit selection
    uint16_t * data2,   // A pointer to the data storage location
    uint8_t * data3     // A pointer to the data storage location
);
```

Description

Read any of the timer unit's counter registers

[data1]
The unit number n (where n = 0).

[data2]
A pointer to where the Timer subcounter register (TCNTS) value shall be stored.
Specify PDL_NO_PTR if it is not required.

[data3]
Where the Timer Interrupt Skipping Counter register (TITCNT) value shall be stored.
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

None.

Remarks

- None.

Program example

```
/* RPDL definitions */
#include "r_pdl_mtu2.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 unit 0 */
    R_MTU2_ReadUnit(
        0,
        &Sub_count,
        &Skip_count
    );
}
```

4.2.15. Port Output Enable

1) R_POE_Set

Synopsis Configure the Port Output Enable module.

Prototype

```
bool R_POE_Set(
        uint32_t data1,    // Input configuration selection
        uint16_t data2,    // Input POEn# pin selection
        uint16_t data3     // Output configuration selection
);
```

Description (1/2) Initialise the POE pins.

[data1]
Configure the input pin detection for pins POE0 to POE3 and POE8.
If multiple selections are required, use "|" to separate each selection.
All settings 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 each pin POE0 to POE3 and POE8, select falling edge or low level for 16 samples at PCLKB ÷ 8, 16 or 128.
PDL_POE_1_MODE_EDGE or PDL_POE_1_MODE_LOW_8 or PDL_POE_1_MODE_LOW_16 or PDL_POE_1_MODE_LOW_128	
PDL_POE_2_MODE_EDGE or PDL_POE_2_MODE_LOW_8 or PDL_POE_2_MODE_LOW_16 or PDL_POE_2_MODE_LOW_128	
PDL_POE_3_MODE_EDGE or PDL_POE_3_MODE_LOW_8 or PDL_POE_3_MODE_LOW_16 or PDL_POE_3_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	

[data2]
Allocate the pins for signals POE0 to POE3 and POE8.
If multiple selections are required, use "|" to separate each selection.
All settings are optional. Specify PDL_NO_DATA if none are required.

PDL_POE_0_PORT_C_4 or PDL_POE_0_PORT_D_7	Pin POE0 input selection
PDL_POE_1_PORT_B_5 or PDL_POE_1_PORT_D_6	Pin POE1 input selection
PDL_POE_2_PORT_3_4 or PDL_POE_2_PORT_A_6 or PDL_POE_2_PORT_D_5	Pin POE2 input selection
PDL_POE_3_PORT_3_3 or PDL_POE_3_PORT_B_3 or PDL_POE_3_PORT_D_4	Pin POE3 input selection
PDL_POE_8_PORT_1_7 or PDL_POE_8_PORT_3_0 or PDL_POE_8_PORT_D_3 or PDL_POE_8_PORT_E_3	Pin POE8 input selection

Description (2/2)

[data3]
Configure pin output control.
If multiple selections are required, use "|" to separate each selection.
All settings are optional. Specify PDL_NO_DATA if none are required.

- High impedance request detection

PDL_POE_HI_Z_REQ_8_ENABLE	If a request is detected on pin POE8, place the MTU channel 0 I/O pins in the high impedance state.
PDL_POE_HI_Z_REQ_MTIOC0A	Select the MTU channel 0 I/O pins that shall be controlled by the high impedance request, software control or the oscillation stop detection flag.
PDL_POE_HI_Z_REQ_MTIOC0B	
PDL_POE_HI_Z_REQ_MTIOC0C	
PDL_POE_HI_Z_REQ_MTIOC0D	

PDL_POE_HI_Z_REQ_OSTSTE	Select the MTIOC0A, MTIOC0B, MTIOC0C, MTIOC0D, MTIOC3B, MTIOC3D, MTIOC4A, MTIOC4B, MTIOC4C, and MTIOC4D pins in high-impedance on detection that oscillation has stopped.

- Output short detection

PDL_POE_SHORT_3_4_HI_Z	If a short is detected, place the all the selected MTU channel 3 and 4 pins in the high impedance state.
PDL_POE_SHORT_MTIOC4BD_A	Select the MTU channel I/O pin pairs that shall be controlled by the short detection response, software control or the oscillation stop detection flag.
PDL_POE_SHORT_MTIOC4AC_A	
PDL_POE_SHORT_MTIOC3BD_A	

Return value

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

Category

Port Output Enable

Reference

R_POE_GetStatus

Remarks

- Do not select MTU pins that are not used.
- Use R_POE_GetStatus to get the oscillation stop detection flag.

Program example

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

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

void func(void)
{
    /* Configure POE pins 0 and 3 */
    R_POE_Set(
        PDL_POE_0_MODE_EDGE | PDL_POE_3_MODE_LOW_128,
        PDL_POE_0_PORT_D_7 | PDL_POE_3_PORT_D_4,
        PDL_NO_DATA
    );
}
```

2) R_POE_Create

Synopsis

Configure the Port Output Enable event handling.

Prototype

```
bool R_POE_Create(
        uint8_t data1,      // Input configuration selection
        void * func1,       // Callback function
        void * func2,       // Callback function
        uint8_t data2       // Interrupt priority level
);
```

Description

Enable interrupts and register 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

PDL_POE_IRQ_HI_Z_0_3_DISABLE or PDL_POE_IRQ_HI_Z_0_3_ENABLE	Disable or enable an interrupt on detection of any high impedance request on pins POE0 to POE3.

- Output short detection response

PDL_POE_IRQ_SHORT_3_4_DISABLE or PDL_POE_IRQ_SHORT_3_4_ENABLE	Disable or enable an interrupt on detection of a short on any MTU channel 3 or 4 two-phase output pair.

[func1]
The function to be called when an enabled request on pins POE0 to POE3 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 a request on pin POE8 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 and func2.

Return value

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

Category

Port Output Enable

Reference

R_POE_Set, R_POE_GetStatus

Remarks

- Use R_POE_GetStatus to determine the interrupt cause.
- 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_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_3_ENABLE,
        POE0_handler,
        PDL_NO_FUNC,
        0x01
    );
}
```

3) R_POE_Control

Synopsis

Control the Port Output Enable module.

Prototype

```
bool R_POE_Control (
        uint8_t data1,      // Control options
        uint16_t data2,     // Control options
        uint8_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 channel high impedance control

PDL_POE_MTU3_MTU4_HI_Z_ON or PDL_POE_MTU3_MTU4_HI_Z_OFF	Control the high impedance state of the MTU3 and MTU4 outputs.
PDL_POE_MTU0_HI_Z_ON or PDL_POE_MTU0_HI_Z_OFF	Control the high impedance state of the MTU0 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_POE1_CLEAR	
PDL_POE_FLAG_POE2_CLEAR	
PDL_POE_FLAG_POE3_CLEAR	
PDL_POE_FLAG_POE8_CLEAR	
PDL_POE_FLAG_OSTSTF_CLEAR	
PDL_POE_FLAG_SHORT_3_4_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_3_DISABLE	Control interrupts on detection of any high impedance request on pins POE0 to POE3.
PDL_POE_IRQ_HI_Z_0_3_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	

- Output short detection response

PDL_POE_IRQ_SHORT_3_4_DISABLE	Control interrupts on detection of a short on any MTU channel 3 or 4 two-phase output pair.
PDL_POE_IRQ_SHORT_3_4_ENABLE	

Return value

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

Category

Port Output Enable

Reference

Remarks

- Call R_POE_Create before using this function.
- Clearing a level-triggered event flag will fail if the trigger is still asserted.
- 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.

Program example

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

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

void func(void)
{
    /* Select high impedance on the MTU0 I/O pins */
    R_POE_Control(
        PDL_POE_MTU0_HI_Z_ON,
        PDL_NO_DATA,
        PDL_NO_DATA
    );
}
```

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 – b10	b9	b8
0	Output short detection MTU3 or MTU4 0: Not detected 1: Detected	0	High impedance request detection (more) OSTSTF 0: No request 1: Requested	POE8

b7	b6	b5	b4	b3	b2	b1	b0
0				High impedance request detection on pin POEn POE3 0: No request 1: Requested	POE2	POE1	POE0

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
    );
}
```

4.2.16. 8-bit Timer

1) R_TMR_Set

Synopsis

Configure the optional TMR pins.

Prototype

```
bool R_TMR_Set(
    uint8_t data1,      // Channel selection
    uint32_t data2      // Configuration
);
```

Description

Set up the global TMR options.

[data1]
The channel number n (where n = 0, 1, 2 or 3).

[data2]
Configure the TMR input and output pins for the channel. Use "|" to separate each selection.

- Valid when n = 0

PDL_TMR_TMR0_TMO0_P22 or PDL_TMR_TMR0_TMO0_PB3 or PDL_TMR_TMR0_TMO0_PH1	Select the pins for TMO0
PDL_TMR_TMR0_TMCI0_P21 or PDL_TMR_TMR0_TMCI0_PB1 or PDL_TMR_TMR0_TMCI0_PH3	Select the pins for TMCI0
PDL_TMR_TMR0_TMRI0_P20 or PDL_TMR_TMR0_TMRI0_PA4 or PDL_TMR_TMR0_TMRI0_PH2	Select the pins for TMRI0

- Valid when n = 1

PDL_TMR_TMR1_TMO1_P17 or PDL_TMR_TMR1_TMO1_P26	Select the pins for TMO1
PDL_TMR_TMR1_TMCI1_P12 or PDL_TMR_TMR1_TMCI1_P54 or PDL_TMR_TMR1_TMCI1_PC4	Select the pins for TMCI1
PDL_TMR_TMR1_TMRI1_P24 or PDL_TMR_TMR1_TMRI1_PB5	Select the pins for TMRI1

- Valid when n = 2

PDL_TMR_TMR2_TMO2_P16 or PDL_TMR_TMR2_TMO2_PC7	Select the pins for TMO2
PDL_TMR_TMR2_TMCI2_P15 or PDL_TMR_TMR2_TMCI2_P31 or PDL_TMR_TMR2_TMCI2_PC6	Select the pins for TMCI2
PDL_TMR_TMR2_TMRI2_P14 or PDL_TMR_TMR2_TMRI2_PC5	Select the pins for TMRI2

- Valid when n = 3

PDL_TMR_TMR3_TMO3_P13 or PDL_TMR_TMR3_TMO3_P32 or PDL_TMR_TMR3_TMO3_P55	Select the pins for TMO3
PDL_TMR_TMR3_TMCI3_P27 or PDL_TMR_TMR3_TMCI3_P34 or PDL_TMR_TMR3_TMCI3_PA6	Select the pins for TMCI3
PDL_TMR_TMR3_TMRI3_P30 or PDL_TMR_TMR3_TMRI3_P33	Select the pins for TMRI3

Return value

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

Category

Timer TMR

Reference

R_TMR_CreateChannel, R_TMR_CreateUnit

Remarks

- Before calling any R_TMR_Create function, call this function to configure the relevant pins.
- Call this function multiple times, if more than one channel is to be configured.
- Pins which are not used for the TMR functions may be omitted.
- The configurations are based on the 100-Pin package device. Some pin options are not available on smaller device packages. Please refer to the "Multifunction Pin Controller (MPC)" section in the RX220 Hardware Manual for details of pin selection.

Program example

```
#include "r_pdl_tmr.h"

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

void func(void)
{
    /* Configure the applicable TMR pins */
    R_TMR_Set(
        0,
        PDL_TMR_TMR0_TMO0_PB3 | PDL_TMR_TMRO_TMCI0_PB1 | \
        PDL_TMR_TMR0_TMRI0_PA4
    );
}
```

2) R_TMR_CreateChannel

Synopsis

Configure a timer TMR channel.

Prototype

```
bool R_TMR_CreateChannel(
    uint8_t data1,      // Channel selection
    uint32_t data2,     // Configuration selection
    uint8_t data3,      // Output control
    uint8_t data4,      // Counter register value
    uint8_t data5,      // Compare match A register value
    uint8_t data6,      // Compare match B register value
    void * func1,       // Overflow callback function
    void * func2,       // Compare match A callback function
    void * func3,       // Compare match B callback function
    uint8_t data7       // Interrupt priority level
);
```

Description (1/2)

Set up an 8-bit timer TMR channel.

[data1]
The channel number n (where n = 0, 1, 2 or 3).

[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.

- Counter clock source selection

PDL_TMR_CLK_OFF or	The clock input is disabled.
PDL_TMR_CLK_EXT_RISING or PDL_TMR_CLK_EXT_FALLING or PDL_TMR_CLK_EXT_BOTH or	The external clock signal TMCIn is used. Select rising, falling or both edges detected.
PDL_TMR_CLK_PCLK_DIV_1 or PDL_TMR_CLK_PCLK_DIV_2 or PDL_TMR_CLK_PCLK_DIV_8 or PDL_TMR_CLK_PCLK_DIV_32 or PDL_TMR_CLK_PCLK_DIV_64 or PDL_TMR_CLK_PCLK_DIV_1024 or PDL_TMR_CLK_PCLK_DIV_8192 or	The internal clock signal PCLKB ÷ 1, 2, 8, 32, 64, 1024 or 8192.
PDL_TMR_CLK_TMR1_OVERFLOW or PDL_TMR_CLK_TMR3_OVERFLOW or	The overflow signal from TMR (n+1). Valid for n = 0 or 2.
PDL_TMR_CLK_TMR0_CM_A or PDL_TMR_CLK_TMR2_CM_A	The compare match A signal from TMR(n-1). Valid for n = 1 or 3.

- Counter clearing

PDL_TMR_CLEAR_DISABLE or	Clearing is disabled.
PDL_TMR_CLEAR_CM_A or	Cleared after a compare match A occurs.
PDL_TMR_CLEAR_CM_B or	Cleared after a compare match B occurs.
PDL_TMR_CLEAR_RESET_RISING or	Cleared by a rising edge on the external reset pin TMRIn.
PDL_TMR_CLEAR_RESET_HIGH	Cleared when the external reset pin TMRIn is high.

- Compare Match A DTC trigger control

PDL_TMR_CM_A_DTC_TRIGGER_DISABLE or PDL_TMR_CM_A_DTC_TRIGGER_ENABLE	Disable or enable activation of the DTC when a Compare Match A occurs.

- Compare Match B DTC trigger control

PDL_TMR_CM_B_DTC_TRIGGER_DISABLE or PDL_TMR_CM_B_DTC_TRIGGER_ENABLE	Disable or enable activation of the DTC when a Compare Match B occurs.

Description (2/2)

[data3]
Configure the output control. 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 control for pin TMOn

PDL_TMR_OUTPUT_IGNORE_CM_A or PDL_TMR_OUTPUT_LOW_CM_A or PDL_TMR_OUTPUT_HIGH_CM_A or PDL_TMR_OUTPUT_INV_CM_A	No change if a compare match A occurs. 0 is output if a compare match A occurs. 1 is output if a compare match A occurs. The output toggles if a compare match A occurs.
PDL_TMR_OUTPUT_IGNORE_CM_B or PDL_TMR_OUTPUT_LOW_CM_B or PDL_TMR_OUTPUT_HIGH_CM_B or PDL_TMR_OUTPUT_INV_CM_B	No change if a compare match B occurs. 0 is output if a compare match B occurs. 1 is output if a compare match B occurs. The output toggles if a compare match B occurs.

[data4]
The counter value.

[data5]
The compare match A value.

[data6]
The compare match B value.

[func1]
The function to be called when an overflow occurs.
Use PDL_NO_FUNC if not required.

[func2]
The function to be called when a Compare match A occurs.
Use PDL_NO_FUNC if not required.

[func3]
The function to be called when a Compare match B occurs.
Use PDL_NO_FUNC if not required.

[data7]
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, func2 and func3.

Return value

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

Category

Timer TMR

Reference

R_TMR_Set

Remarks

- Please use R_TMR_Set to select the input (TMCIn, TMRIn) and output (TMOn) pins as required. This function will return false if a pin is enabled but is not set properly.
- A closed clock loop will be created if:
  The overflow signal from TMR1 is selected for TMR0 and the compare match A signal from TMR0 is selected for TMR1, or
  The overflow signal from TMR3 is selected for TMR2 and the compare match A signal from TMR2 is selected for TMR3.
  Either case should be avoided.
- The output will be high-impedance when PDL_TMR_OUTPUT_IGNORE_CM_A and PDL_TMR_OUTPUT_IGNORE_CM_B are selected.
- If a callback function is specified, this function will enable the relevant interrupt.
  Please see the notes on callback function usage 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.

Program example

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

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

void func(void)
{
    /* Configure TMR0: PCLKB, clear after a compare match A */
    R_TMR_CreateChannel(
        0,
        PDL_TMR_CLK_PCLK_DIV_1 | PDL_TMR_CLEAR_CM_A,
        PDL_NO_DATA,
        0,
        199,
        99,
        PDL_NO_FUNC,
        PDL_NO_FUNC,
        PDL_NO_FUNC,
        0
    );
}
```

3) R_TMR_CreateUnit

Synopsis

Configure a timer TMR unit.

Prototype

```
bool R_TMR_CreateUnit(
    uint8_t data1,      // Unit selection
    uint32_t data2,     // Configuration selection
    uint8_t data3,      // Output control
    uint16_t data4,     // 16-bit counter register value
    uint16_t data5,     // 16-bit compare match A register value
    uint16_t data6,     // 16-bit compare match B register value
    void * func1,       // Overflow callback function
    void * func2,       // Compare match A callback function
    void * func3,       // Compare match B callback function
    uint8_t data7       // Interrupt priority level
);
```

Description (1/2)

Set up a timer TMR unit in 16-bit count mode.

[data1]
The unit number n (where n = 0 or 1).

[data2]
Configure the unit. 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.

- Counter clock source selection

PDL_TMR_CLK_OFF or	The clock input is disabled.
PDL_TMR_CLK_EXT_RISING or PDL_TMR_CLK_EXT_FALLING or PDL_TMR_CLK_EXT_BOTH or	The external clock signal TMCIx (x = 1 or 3 for n = 0 or 1) is used, with rising, falling or both edges detected.
PDL_TMR_CLK_PCLK_DIV_1 or PDL_TMR_CLK_PCLK_DIV_2 or PDL_TMR_CLK_PCLK_DIV_8 or PDL_TMR_CLK_PCLK_DIV_32 or PDL_TMR_CLK_PCLK_DIV_64 or PDL_TMR_CLK_PCLK_DIV_1024 or PDL_TMR_CLK_PCLK_DIV_8192	The internal clock signal PCLKB ÷ 1, 2, 8, 32, 64, 1024 or 8192.

- Counter clearing

PDL_TMR_CLEAR_DISABLE or	Clearing is disabled.
PDL_TMR_CLEAR_CM_A or	Cleared after a compare match A occurs.
PDL_TMR_CLEAR_CM_B or	Cleared after a compare match B occurs.
PDL_TMR_CLEAR_RESET_RISING or	Cleared by a rising edge on the external reset pin TMRIn.
PDL_TMR_CLEAR_RESET_HIGH	Cleared when the external reset pin TMRIx (x = 0 or 2 for n = 0 or 1) is high.

- Compare Match A DTC trigger control

PDL_TMR_CM_A_DTC_TRIGGER_DISABLE or PDL_TMR_CM_A_DTC_TRIGGER_ENABLE	Disable or enable activation of the DTC when a Compare Match A occurs.

- Compare Match B DTC trigger control

PDL_TMR_CM_B_DTC_TRIGGER_DISABLE or PDL_TMR_CM_B_DTC_TRIGGER_ENABLE	Disable or enable activation of the DTC when a Compare Match B occurs.

Description (2/2)

[data3]
Configure the output control. If multiple selections are required, use "|" to separate each selection.
The default settings are shown in **bold**.

- Output control for pin TMOy (y = 0 or 2 for n = 0 or 1)

PDL_TMR_OUTPUT_IGNORE_CM_A or PDL_TMR_OUTPUT_LOW_CM_A or PDL_TMR_OUTPUT_HIGH_CM_A or PDL_TMR_OUTPUT_INV_CM_A	No change if a compare match A occurs. 0 is output if a compare match A occurs. 1 is output if a compare match A occurs. The output toggles if a compare match A occurs.
PDL_TMR_OUTPUT_IGNORE_CM_B or PDL_TMR_OUTPUT_LOW_CM_B or PDL_TMR_OUTPUT_HIGH_CM_B or PDL_TMR_OUTPUT_INV_CM_B	No change if a compare match B occurs. 0 is output if a compare match B occurs. 1 is output if a compare match B occurs. The output toggles if a compare match B occurs.

[data4]
The 16-bit counter value.

[data5]
The 16-bit compare match A value.

[data6]
The 16-bit compare match B value.

[func1]
The function to be called when an overflow occurs.
Use PDL_NO_FUNC if not required.

[func2]
The function to be called when a Compare match A occurs.
Use PDL_NO_FUNC if not required.

[func3]
The function to be called when a Compare match B occurs.
Use PDL_NO_FUNC if not required.

[data7]
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, func2 and func3.

Return value

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

Category

Timer TMR

Reference

R_TMR_Set

Remarks

- Please use R_TMR_Set to select the input (TMCIn, TMRIn) and output (TMOn) pins as required. This function will return false if a pin is enabled but is not set properly.
- The output will be high-impedance when PDL_TMR_OUTPUT_IGNORE_CM_A and PDL_TMR_OUTPUT_IGNORE_CM_B are selected.
- If a callback function is specified, this function will enable the relevant interrupt. Please see the notes on callback function usage 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.

Program example

```
#include "r_pdl_tmr.h"

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

void func(void)
{
    /* Configure TMR unit 0: PCLKB, clear after a compare match A */
    R_TMR_CreateUnit(
        0,
        PDL_TMR_CLK_PCLK_DIV_1 | PDL_TMR_CLEAR_CM_A,
        0,
        0,
        199,
        99,
        PDL_NO_FUNC,
        PDL_NO_FUNC,
        PDL_NO_FUNC,
        0
    );
}
```

4) R_TMR_CreatePeriodic

Synopsis

Select periodic operation.

Prototype

```
bool R_TMR_CreatePeriodic(
    uint8_t data1,     // 8-bit (channel) or 16-bit (unit) selection
    uint32_t data2,    // Configuration selection
    double data3,      // Period or frequency
    double data4,      // Pulse width or duty cycle
    void * func1,      // Pulse width interval callback function
    void * func2,      // Periodic interval callback function
    uint8_t data5      // Interrupt priority level
);
```

Description

Set up a TMR timer channel or unit for periodic operation and start the timer.

[data1]

PDL_TMR_TMR0 or PDL_TMR_TMR1 or PDL_TMR_TMR2 or PDL_TMR_TMR3 or PDL_TMR_UNIT0 or PDL_TMR_UNIT1	The channel n (n = 0, 1, 2 or 3) or unit (n = 0 or 1) to be configured.

[data2]

Configure the timer. If multiple selections are required, use "|" to separate each selection.
The default settings are shown in **bold**.

- Period or frequency calculation

PDL_TMR_PERIOD or PDL_TMR_FREQUENCY	The parameters data3 and data4 will contain either period and pulse width or frequency and duty cycle.

- Output pin control

PDL_TMR_OUTPUT_HIGH or PDL_TMR_OUTPUT_LOW or **PDL_TMR_OUTPUT_OFF**	Start with a high-level or low-level output, or no output on pin TMOn. For 16-bit operation the pin shall be TMO2 when n = 1.

- Pulse DTC trigger control

PDL_TMR_PULSE_DTC_TRIGGER_DISABLE or PDL_TMR_PULSE_DTC_TRIGGER_ENABLE	Disable or enable activation of the DTC at the pulse width interval.

- Period DTC trigger control

PDL_TMR_PERIOD_DTC_TRIGGER_DISABLE or PDL_TMR_PERIOD_DTC_TRIGGER_ENABLE	Disable or enable activation of the DTC at the periodic interval.

[data3]

The period (in seconds) or frequency (in Hz).

[data4]

The pulse width (in seconds) or duty cycle (%).

[func1]

The function to be called at the pulse width interval. Use PDL_NO_FUNC if not required.

[func2]

The function to be called at the periodic interval. Use PDL_NO_FUNC if not required.

[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 both parameters func1and func2.

Return value

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

Category

Timer TMR

Reference

R_CGC_Set, R_TMR_CreateChannel, R_TMR_CreateUnit

Remarks

- Function R_CGC_Set must be called (with the current clock source selected) before using this function.
- This function is an alternative to R_TMR_CreateChannel and R_TMR_CreateUnit.
- Please use R_TMR_Set to select the output (TMOn) pin as required. This function will return false if a pin is enabled but is not set properly.
- If a callback function is specified, this function will enable the relevant interrupt. Please see the notes on callback function use in §6.
- The timing limits depend on the peripheral module clock, PCLKB.

		f_{PCLKB} (MHz)		
	Equation	12.5	12	8
Timer resolution	$\frac{1}{f_{PCLKB}}$	80ns	83.3ns	125ns
$Period_{MIN}$	$\frac{2}{f_{PCLKB}}$	160ns	166.7ns	250ns
$Period_{MAX\_CHANNEL}$	$\frac{2^{21}}{f_{PCLKB}}$	167.7ms	174.8ms	262ms
$Period_{MAX\_UNIT}$	$\frac{2^{29}}{f_{PCLKB}}$	42.9s	44.7s	67.1s
$Width_{MIN}$	$Period_{MIN}$			
$Width_{MAX\_CHANNEL}$	$Period_{MAX\_CHANNEL}$			
$Width_{MAX\_UNIT}$	$Period_{MAX\_UNIT}$			
f_{MAX}	$\frac{f_{PCLKB}}{2}$	6.25 MHz	6 MHz	4 MHz
$f_{MIN\_CHANNEL}$	$\frac{f_{PCLKB}}{2^{21}}$	5.96 Hz	5.7 Hz	3.81 Hz
f_{MIN_UNIT}	$\frac{f_{PCLKB}}{2^{29}}$	0.0232 Hz	0.0224 Hz	0.0149 Hz

- If the requested period is not a multiple of the timer resolution, the actual time period will be more than the requested time period.
- The actual duty cycle will be less than the requested duty cycle if the resulting pulse width is not a multiple of the timer resolution.
- 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_tmr.h"

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

void func(void)
{
    /* Configure pin TMO1 for 500ns period, 200ns pulse width */
    R_TMR_CreatePeriodic(
        PDL_TMR_TMR1,
        PDL_TMR_PERIOD | PDL_TMR_OUTPUT_HIGH,
        500E-9,
        200E-9,
        PDL_NO_FUNC,
        PDL_NO_FUNC,
        0
    );

    /* Configure pin TMO1 for 5MHz frequency, 60% duty cycle */
    R_TMR_CreatePeriodic(
        PDL_TMR_TMR1,
        PDL_TMR_FREQUENCY | PDL_TMR_OUTPUT_HIGH,
        5E6,
        60,
        PDL_NO_FUNC,
        PDL_NO_FUNC,
        0
    );
}
```

5) R_TMR_CreateOneShot

Synopsis

Configure and use a one-shot timer.

Prototype

```
bool R_TMR_CreateOneShot(
    uint8_t data1,      // 8-bit (channel) or 16-bit (unit) timer selection
    uint32_t data2,     // Configuration selection
    double data3,       // Period
    void * func,        // Callback function
    uint8_t data4       // Interrupt priority level
);
```

Description

Set up a TMR timer channel or unit for one-shot operation and start the timer.

[data1]

PDL_TMR_TMR0 or PDL_TMR_TMR1 or PDL_TMR_TMR2 or PDL_TMR_TMR3 or PDL_TMR_UNIT0 or PDL_TMR_UNIT1	The channel n (n = 0, 1, 2 or 3) or unit n (n = 0 or 1) to be configured.

[data2]

Configure the timer. Use "|" to separate each selection. The default settings are shown in **bold**. Specify PDL_NO_DATA to use the defaults.

- Output pin control

PDL_TMR_OUTPUT_HIGH or PDL_TMR_OUTPUT_LOW or **PDL_TMR_OUTPUT_OFF**	For the duration of the one-shot period, generate a high-level output, low-level output or no output on pin TMOn. For 16-bit operation the pin shall be TMO2 when n = 1.

- DTC trigger control

PDL_TMR_PULSE_DTC_TRIGGER_DISABLE or PDL_TMR_PULSE_DTC_TRIGGER_ENABLE	Disable or enable activation of the DTC when the one-shot period ends.

- Control the CPU during the one-shot operation.

PDL_TMR_CPU_ON or	Allow the CPU to run normally while the one-shot operates.
PDL_TMR_CPU_OFF	Stop the CPU when the one-shot timer starts. The CPU will re-start when any valid interrupt occurs.

[data3]

The one-shot time period (in seconds).

[func]

The function to be called when the one-shot period ends.
Specify PDL_NO_FUNC for this function to wait for the timer to complete before returning.
You should always specify a function if PDL_TMR_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

Timer TMR

Reference

R_CGC_Set, R_TMR_CreateChannel, R_TMR_CreateUnit

Remarks

- Function R_CGC_Set must be called (with the current clock source selected) before using this function.
- This function is an alternative to R_TMR_CreateChannel and R_TMR_CreateUnit.
- Please use R_TMR_Set to select the output (TMOn) pin as required. This function will return false if a pin is enabled but is not set properly.
- This function stops the timer on completion, so no other TMR function calls are required.
- If a callback function is specified, this function will enable the relevant interrupt. Please see the notes on callback function usage in §6.
- If no callback function is specified, this function waits for the CMIB flag to indicate that the one-shot time delay is complete. If the timer's control registers are directly modified by the user, this function may lock up.
- 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 timer period limits depend on the peripheral module clock, PCLKB.

		f_{PCLKB} (MHz)		
	Equation	12.5	12	8
T_{MIN}	$\frac{1}{f_{PCLKB}}$	80ns	83.3ns	125ns
$T_{MAX\_CHANNEL}$	$\frac{2^{21}}{f_{PCLKB}}$	167.7ms	174.8ms	262ms
T_{MAX_UNIT}	$\frac{2^{29}}{f_{PCLKB}}$	42.9s	44.7s	67.1s

Program example

```
#include "r_pdl_tmr.h"

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

void func(void)
{
    /* Output a pulse and wait for 40ms */
    R_TMR_CreateOneShot(
        PDL_TMR_TMR0,
        PDL_TMR_OUTPUT_HIGH,
        40E-3,
        PDL_NO_FUNC,
        0
    );
}
```

6) R_TMR_Destroy

Synopsis

Disable a TMR timer unit.

Prototype

```
bool R_TMR_Destroy(
    uint8_t data    // Unit selection
);
```

Description

Shut down a TMR timer unit.

[data]
The timer unit n (where n = 0 or 1).
Unit 0 comprises channels TMR0 and TMR1.
Unit 1 comprises channels TMR2 and TMR3.

Return value

True.

Category

Timer TMR

Reference

None.

Remarks

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

Program example

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

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

void func(void)
{
    /* Shutdown channels 0 and 1 */
    R_TMR_Destroy(
        0
    );
}
```

7) R_TMR_ControlChannel

Synopsis

Write to timer channel registers.

Prototype

```
bool R_TMR_ControlChannel(
    uint8_t data1,      // Channel selection
    uint32_t data2,     // Configuration selection
    uint8_t data3,      // Counter register value
    uint8_t data4,      // Compare match A register value
    uint8_t data5       // Compare match B register value
);
```

Description

Modify a timer channel's operation, counter and compare registers.

[data1]
The channel number n (where n = 0, 1, 2 or 3).

[data2]
The channel settings to be modified.
If multiple selections are required, use "|" to separate each selection.

- Counter stop / re-start

PDL_TMR_STOP or PDL_TMR_START	Disable or re-enable the counter clock source.

- The counter or compare registers to be modified.

PDL_TMR_COUNTER	Update the timer counter register (TCNT).
PDL_TMR_TIME_CONSTANT_A	Update the timer compare match A register (TCORA).
PDL_TMR_TIME_CONSTANT_B	Update the timer compare match B register (TCORB).

- Counter stop in response to ELC

PDL_TMR_ELC_COUNT_STOP	Stop a counter that was started by the ELC.

[data3]
The counter value. This will be ignored if the register is not selected.

[data4]
The compare match A value. This will be ignored if the register is not selected.

[data5]
The compare match B value. This will be ignored if the register is not selected.

Return value

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

Category

Timer TMR

Reference

R_TMR_CreateChannel

Remarks

- PDL_TMR_STOP is to disable the counter clock source, but PDL_TMR_ELC_COUNT_STOP is to stop the counter triggered by the ELC. If PDL_TMR_ELC_COUNT_STOP is selected, the counter is stopped, but the clock source is still running. The system will wait for the next ELC event to trigger the counter again.

Program example

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

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

void func(void)
{
    /* Load the counter on channel TMR0 */
    R_TMR_ControlChannel(
        0,
        PDL_TMR_COUNTER,
        0xFF,
        PDL_NO_DATA,
        PDL_NO_DATA
    );
}
```

8) R_TMR_ControlUnit

Synopsis

Write to timer unit registers.

Prototype

```
bool R_TMR_ControlUnit(
    uint8_t data1,     // Unit selection
    uint32_t data2,    // Configuration selection
    uint16_t data3,    // 16-bit counter register value
    uint16_t data4,    // 16-bit compare match A register value
    uint16_t data5     // 16-bit compare match B register value
);
```

Description

Modify a timer unit's counter and compare registers.

[data1]
The unit number n (where n = 0 or 1).

[data2]
The channel settings to be modified.
If multiple selections are required, use "|" to separate each selection.

- Counter stop / re-start

PDL_TMR_STOP or PDL_TMR_START	Disable or re-enable the counter clock source.

- The counter or compare registers to be modified.

PDL_TMR_COUNTER	Update the timer counter register (TCNT).
PDL_TMR_TIME_CONSTANT_A	Update the timer compare match A register (TCORA).
PDL_TMR_TIME_CONSTANT_B	Update the timer compare match B register (TCORB).

[data3]
The 16-bit counter value. This will be ignored if the register is not selected.

[data4]
The 16-bit compare match A value. This will be ignored if the register is not selected.

[data5]
The 16-bit compare match B value. This will be ignored if the register is not selected.

Return value

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

Category

Timer TMR

Reference

R_TMR_CreateUnit

Remarks

- For unit 0, the upper byte is the value for TMR0 and the lower byte is the value for TMR1.
  For unit 1, the upper byte is the value for TMR2 and the lower byte is the value for TMR3.

Program example

```
/* RPDL definitions */
#include "r_pdl_tmr.h”

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

void func(void)
{
    /* Load the unit 1 counter and constants */
    R_TMR_ControlUnit(
        1,
        PDL_TMR_COUNTER | PDL_TMR_TIME_CONSTANT_A | \
        PDL_TMR_TIME_CONSTANT_B,
        0xAAFF,
        0x100,
        0x5600
    );
}
```

9) R_TMR_ControlPeriodic

Synopsis

Control periodic operation.

Prototype

```
bool R_TMR_ControlPeriodic(
    uint8_t data1,      // 8-bit (channel) or 16-bit (unit) selection
    uint32_t data2,     // Configuration selection
    double data3,       // The new period or frequency
    double data4        // The new pulse width or duty cycle
);
```

Description

Modify a periodic timer operation.

[data1]

PDL_TMR_TMR0 or PDL_TMR_TMR1 or PDL_TMR_TMR2 or PDL_TMR_TMR3 or PDL_TMR_UNIT0 or PDL_TMR_UNIT 1	The channel n (n = 0, 1, 2 or 3) or unit (n = 0 or 1) to be configured.

[data2]

Select the options to be modified. Use "|" to separate each selection.

- Period or frequency calculation

PDL_TMR_PERIOD or PDL_TMR_FREQUENCY	The parameters data3 and data4 will contain either period and pulse width or frequency and duty cycle.

- Output pin control

PDL_TMR_OUTPUT_ENABLE or PDL_TMR_OUTPUT_DISABLE	Enable or disable the periodic output on pin TMOn. For 16-bit operation the pin shall be TMO2 when n = 1.

- Counter stop / start

PDL_TMR_STOP or PDL_TMR_START	Disable or re-enable the counter clock source.

[data3]

The new period or frequency. This will be ignored if a timing change is not requested.

[data4]

The new pulse width or duty cycle (%). This will be ignored if a timing change is not requested.

Return value

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

Category

Timer TMR

Reference

R_TMR_CreatePeriodic

Remarks

- See the remarks for R_TMR_CreatePeriodic.

Program example

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

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

void func(void)
{
    /* Change timer TMR1 to 600ns period, 100ns pulse width */
    R_TMR_ControlPeriodic(
        PDL_TMR_TMR1,
        PDL_TMR_PERIOD,
        600E-9,
        100E-9
    );
}
```

10) R_TMR_ReadChannel

Synopsis

Read from timer channel registers.

Prototype

```
bool R_TMR_ReadChannel(
    uint8_tdata1,      // Channel selection
    uint8_t * data2,   // A pointer to the data storage location
    uint8_t * data3,   // A pointer to the data storage location
    uint8_t * data4,   // A pointer to the data storage location
    uint8_t * data5    // A pointer to the data storage location
);
```

Description

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

[data1]
The channel number n (where n = 0, 1, 2 or 3).

[data2]
The status flags shall be stored in the format below.
The flag will be set to 1 if the condition has been detected.
Specify PDL_NO_PTR if the flags are not to be read.

b7 – b4	b3	b2	b1	b0
0	ELC count state	Overflow	Compare match B	Compare match A

[data3]
A pointer to where the counter value shall be stored. Specify PDL_NO_PTR if it is not required.

[data4]
Where the compare match A value shall be stored. Specify PDL_NO_PTR if it is not required.

[data5]
Where the compare match B value shall be stored. Specify PDL_NO_PTR if it is not required.

Return value

True.

Category

Timer TMR

Reference

R_TMR_CreateChannel

Remarks

- If the status flags are read, any flag that has been set to 1 shall be automatically cleared to 0 by this function.
- The ELC count flag is valid only when n = 0 or 2.

Program example

```
#include "r_pdl_tmr.h"

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

uint8_t Flags;
uint8_t Counter;
uint8_t CompareMatchA;
uint8_t CompareMatchB;

void func(void)
{
    /* Read the status flags and registers for TMR0 */
    R_TMR_ReadChannel(
        0,
        &Flags,
        &Counter,
        &CompareMatchA,
        &CompareMatchB
    );
}
```

11) R_TMR_ReadUnit

Synopsis

Read from timer unit registers.

Prototype

```
bool R_TMR_ReadUnit(
    uint8_t data1,      // Unit 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
);
```

Description

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

[data1]
The unit number n (where n = 0 or 1).

[data2]
The status flags shall be stored in the format below.
A flag will be set to 1 if the condition has been detected.
Specify PDL_NO_PTR if the flags are not to be read.

The unit 0 status flags shall be stored in the format:

b7	b6	b5	b4	b3	b2	b1	b0
0	TMR0			0	TMR1		
	Overflow	Compare match B	Compare match A		Overflow	Compare match B	Compare match A

The unit 1 status flags shall be stored in the format:

b7	b6	b5	b4	b3	b2	b1	b0
0	TMR2			0	TMR3		
	Overflow	Compare match B	Compare match A		Overflow	Compare match B	Compare match A

[data3]
Where the counter value shall be stored. Specify PDL_NO_PTR if it is not required.

[data4]
Where the compare match A value shall be stored. Specify PDL_NO_PTR if it is not required.

[data5]
Where the compare match B value shall be stored. Specify PDL_NO_PTR if it is not required.

Return value

True.

Category

Timer TMR

Reference

R_TMR_CreateUnit

Remarks

- If the status 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_tmr.h"

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

uint8_t Flags;
uint16_t Counter;
uint16_t CompareMatchA;
uint16_t CompareMatchB;

void func(void)
{
    /* Read the status flags and registers for TMR unit 0 */
    R_TMR_ReadUnit(
        0,
        &Flags,
        &Counter,
        &CompareMatchA,
        &CompareMatchB
    );
}
```

4.2.17. 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
    double data3,       // Period, frequency or register data
    void * func,        // Callback function
    uint8_t data4       // Interrupt priority level
);
```

Description

Set up a Compare Match Timer channel.

[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 PCLKB ÷ 8, 32, 128 or 512 as the counter clock source. The parameter data3 will be the register CMCOR value.

- Counter start control

PDL_CMT_START or PDL_CMT_STOP	Enable or disable the starting of the timer count operation.

- DMAC / DTC trigger control

PDL_CMT_DMAC_DTC_TRIGGER_DISABLE or PDL_CMT_DMAC_TRIGGER_ENABLE or PDL_CMT_DTC_TRIGGER_ENABLE	Disable or enable activation of the DMAC or DTC when a compare match occurs.

[data3]
The data to be used for the register value calculations.

Data use	Parameter type
The timer period in seconds or	double
The timer frequency in Hz or	double
The value to be put in register CMCOR	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_CGC_Set

Remarks

- Function R_CGC_Set must be called (with the current clock source selected) before using this function.
- 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, PCLKB.

	Equation	f_{PCLKB} (MHz)			
		32	12.5	12	8
$Period_{MIN}$	$\frac{8}{f_{PCLKB}}$	250ns	640ns	667ns	1.0µs
$Period_{MAX}$	$\frac{2^{25}}{f_{PCLKB}}$	1.05s	2.68s	2.79s	4.19s
f_{MAX}	$\frac{f_{PCLKB}}{8}$	4.0 MHz	1.56 MHz	1.5 MHz	1.0 MHz
f_{MIN}	$\frac{f_{PCLKB}}{2^{25}}$	0.95 Hz	0.37 Hz	0.357 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
    double 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.

PDL_CMT_CPU_ON 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.

- DMAC / DTC trigger control

PDL_CMT_DMAC_DTC_TRIGGER_DISABLE or PDL_CMT_DMAC_TRIGGER_ENABLE or PDL_CMT_DTC_TRIGGER_ENABLE	Disable or enable activation of the DMAC or 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, R_CGC_Control

Remarks

- Function R_CGC_Set must be called (with the current clock source selected) before using 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, PCLKB.

	Equation	f_{PCLKB} (MHz)			
		32	12.5	12	8
T_{MIN}	$\frac{8}{f_{PCLK}}$	250ns	640ns	666.67ns	1µs
T_{MAX}	$\frac{2^{25}}{f_{PCLK}}$	1.05s	2.68s	2.79s	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,
        PDL_NO_DATA,
        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
    double 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 (CMCOR) 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	double
The timer frequency in Hz or	double
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 used first 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.
  To avoid register access conflicts or invalid calls to the callback function, use this method when changing any value.
- If the CMCNT register value is changed to the same value as the CMCOR register, the CMCNT register will be set to 0.

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_STOP | PDL_CMT_PERIOD | PDL_CMT_START,
        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
    );
}
```

4.2.18. Real-time Clock

1) R_RTC_Create

Synopsis

Configure the Real-time clock in calendar count mode.

Prototype

```
bool R_RTC_Create(
    uint32_t data1,     // Configuration selection
    uint32_t data2,     // Current time
    uint32_t data3,     // Current date
    uint16_t data4,     // Periodic configuration
    uint32_t data5,     // Alarm Time
    uint32_t data6,     // Alarm Date
    void * func1,       // Callback function
    uint8_t data7,      // Interrupt priority level
    void * func2,       // Callback function
    uint8_t data8       // Interrupt priority level
);
```

Description (1/3)

Set up and start the Real-time clock in calendar count mode.

[data1]
Configure the clock options.
To set multiple options at the same time, use “|” to separate each value.
The default settings are shown in **bold**. Specify PDL_NO_DATA to use the defaults if not enabling the alarm.

- 12 or 24 hour mode

PDL_RTC_24_HOUR_MODE or PDL_RTC_12_HOUR_MODE	Select 12 or 24 hour mode.

- Alarm enabling

PDL_RTC_ALARM_HOUR_ENABLE	All three can be enabled using: PDL_RTC_ALARM_TIME_ENABLE
PDL_RTC_ALARM_MINUTE_ENABLE	
PDL_RTC_ALARM_SECOND_ENABLE	
PDL_RTC_ALARM_YEAR_ENABLE	All four can be enabled using: PDL_RTC_ALARM_DATE_ENABLE
PDL_RTC_ALARM_MONTH_ENABLE	
PDL_RTC_ALARM_DAY_ENABLE	
PDL_RTC_ALARM_DOW_ENABLE	

- Clock output control

PDL_RTC_OUTPUT_DISABLE or PDL_RTC_OUTPUT_ENABLE	Disable or enable the 1 Hz/64 Hz clock output on the RTCOUT pin.

- Clock RTCOUT output period Select

PDL_RTC_OUTPUT_RTCOS_1HZ or PDL_RTC_OUTPUT_RTCOS_64HZ	RTCOUT outputs 1-Hz RTCOUT outputs 64-Hz

- Configure RTCOUT Pin. Select PDL_NO_DATA if no pins are required.

  PDL_RTC_OUTPUT_ENABLE will not be a valid option if select PDL_NO_DATA

PDL_RTC_PIN_RTCOUT_P16 or PDL_RTC_PIN_RTCOUT_P32	If using the RTCOUT pin then select the port to use for it.

Description (2/3)

[data2]
The current day of the week (DOW) and time in hours, minutes and seconds.
BCD format is used.
The format is dependent upon if using 12 hour or 24 hour mode.

24 Hour Mode:

b31 – b24	b23 – b16	b15 – b8	b7 – b0
Day of week Valid from 0 to 6. 0 = Sunday. Specify 0xFF for automatic calculation using the values in data3.	Hours Valid from 0 to 23.	Minutes Valid from 0 to 59.	Seconds Valid from 0 to 59.

12 Hour Mode:

b31 – b24	b23	b22 – b16	b15 – b8	b7 – b0
Day of week Valid from 0 to 6. 0 = Sunday. Specify 0xFF for automatic calculation using the values in data3.	PM 0 = AM 1 = PM	Hours Valid from 1 to 12.	Minutes Valid from 0 to 59.	Seconds Valid from 0 to 59.

[data3]
The current year, month and day. BCD format is used. If not required, specify PDL_NO_DATA.

b31 – b16	b15 – b8	b7 – b0
Year Valid from 0 to 9999.	Month Valid from 1 to 12.	Day Valid from 1 to the number of days in the month.

[data4]
Configure the clock periodic interrupt.
The default setting is shown in **bold**.

- Periodic interrupt selection

PDL_RTC_PERIODIC_DISABLE or PDL_RTC_PERIODIC_256_HZ or PDL_RTC_PERIODIC_128_HZ or PDL_RTC_PERIODIC_64_HZ or PDL_RTC_PERIODIC_32_HZ or PDL_RTC_PERIODIC_16_HZ or PDL_RTC_PERIODIC_8_HZ or PDL_RTC_PERIODIC_4_HZ or PDL_RTC_PERIODIC_2_HZ or PDL_RTC_PERIODIC_1_HZ or PDL_RTC_PERIODIC_2S	The frequency or interval for periodic interrupt requests.

[data5]
The alarm day of the week and time in hours, minutes and seconds. BCD format is used.
If not required, specify PDL_NO_DATA.
The format is dependent upon if using 12 hour or 24 hour mode.

24 Hour Mode:

b31 – b24	b23 – b16	b15 – b8	b7 – b0
Day of week Valid from 0 to 6. 0 = Sunday. Specify 0xFF for automatic calculation using the values in data5.	Hours Valid from 0 to 23.	Minutes Valid from 0 to 59.	Seconds Valid from 0 to 59.

12 Hour Mode:

b31 – b24	b23	b22 – b16	b15 – b8	b7 – b0
Day of week Valid from 0 to 6. 0 = Sunday. Specify 0xFF for automatic calculation using the values in data3.	PM 0 = AM 1 = PM	Hours Valid from 1 to 12.	Minutes Valid from 0 to 59.	Seconds Valid from 0 to 59.

Description (3/3)

[data6]
The alarm year, month and day. BCD format is used. If not required, specify PDL_NO_DATA.

b31 – b16	b15 – b8	b7 – b0
Year Valid from 0 to 9999.	Month Valid from 1 to 12.	Day Valid from 1 to the number of days in the month.

[func1]
The function to be called when an alarm occurs. Specify PDL_NO_FUNC if not required.

[data7]
The alarm 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 func1.

[func2]
The function to be called at the periodic interval. Specify PDL_NO_FUNC if not required.

[data8]
The periodic 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 func2.

Return value

True if all parameters are valid; otherwise false.

Category

Real-time clock

Reference

R_RTC_Read

Remarks

- The check for days in the month allows for leap years.
- If entering software standby mode soon after starting the RTC, use R_RTC_Read first to confirm that the values are correct.
- The oscillation accuracy of the sub-clock is affected when an on-chip debugger emulator is connected and the sub-clock drive setting is low.
- Before calling this function the count source must be enabled and stable. Refer R_CGC_Set and R_CGC_Control for count source and stabilization time configuration.
- This function is called to use RTC after setting option PDL_CGC_RTC_TO_BE_USED in R_CGC_Control at cold start.
- This function is not required when using 48-pin package.

Program example

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

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

void alarm_function(void){}

void func(void)
{
    /* Configure the clock for an alarm at 12 noon every day */
    /* Using default 24 hour mode. */
    R_RTC_Create(
        PDL_RTC_ALARM_HOUR_ENABLE | \
         PDL_RTC_ALARM_MINUTE_ENABLE |
         PDL_RTC_ALARM_SECOND_ENABLE,
        0xFF114200,    /* Automatic day of week; 11:42:00 */
        0x20100916,    /* 16-Sep-2010 */
        PDL_NO_DATA,
        0x00120000,    /* Alarm at 12 noon */
        PDL_NO_DATA,
        alarm_function,
        15,
        PDL_NO_FUNC,
        PDL_NO_DATA
    );
}
```

2) R_RTC_CreateBinary

Synopsis

Configure the RTC module in binary count mode.

Prototype

```
bool R_RTC_CreateBinary(
    uint32_t data1,     // Configuration selection
    uint32_t data2,     // Current count
    uint16_t data3,     // Periodic configuration
    uint32_t data4,     // Alarm count
    uint32_t data5,     // Alarm mask
    void * func1,       // Callback function
    uint8_t data6,      // Interrupt priority level
    void * func2,       // Callback function
    uint8_t data7       // Interrupt priority level
);
```

Description (1/2)

Set up the RTC operation in Binary count mode

[data1]
Configure the clock options.
To set multiple options at the same time, use "|" to separate each value.
The default settings are shown in **bold**. Specify PDL_NO_DATA to use the defaults.

- Clock output control

PDL_RTC_OUTPUT_DISABLE or PDL_RTC_OUTPUT_ENABLE	Disable or enable the 1 Hz/64 Hz clock output on the RTCOUT pin.

- Clock RTCOUT output period Select

PDL_RTC_OUTPUT_RTCOS_1HZ or PDL_RTC_OUTPUT_RTCOS_64HZ	RTCOUT outputs 1-Hz RTCOUT outputs 64-Hz

- Configure RTCOUT Pin. Select PDL_NO_DATA if no pins are required.

  PDL_RTC_OUTPUT_ENABLE will not be a valid option if select PDL_NO_DATA

PDL_RTC_PIN_RTCOUT_P16 or PDL_RTC_PIN_RTCOUT_P32	If using the RTCOUT pin then select the port to use for it.

[data2]
The current count value. Count seconds in 32 bits, binary display.

[data3]
Configure the clock periodic interrupt.
The default setting is shown in **bold**.

- Periodic interrupt selection

PDL_RTC_PERIODIC_DISABLE or PDL_RTC_PERIODIC_256_HZ or PDL_RTC_PERIODIC_128_HZ or PDL_RTC_PERIODIC_64_HZ or PDL_RTC_PERIODIC_32_HZ or PDL_RTC_PERIODIC_16_HZ or PDL_RTC_PERIODIC_8_HZ or PDL_RTC_PERIODIC_4_HZ or PDL_RTC_PERIODIC_2_HZ or PDL_RTC_PERIODIC_1_HZ or PDL_RTC_PERIODIC_2S	The frequency or interval for periodic interrupt requests.

[data4]
The alarm count value. Setting the alarm register corresponding to 32-bit binary counter.
If not required, specify PDL_NO_DATA.

[data5]
The alarm mask-value in 32 bits binary format.Setting the alarm enable corresponding to 32-bit binary alarm counter.
If not required, specify PDL_NO_DATA.

Description (2/2)

[func1]
The function to be called when an alarm occurs. Specify PDL_NO_FUNC if not required.

[data6]
The alarm 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 func1.

[func2]
The function to be called at the periodic interval. Specify PDL_NO_FUNC if not required.

[data7]
The periodic 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 func2.

Return value

True if all parameters are valid; otherwise false.

Category

Real-time clock

Reference

R_RTC_ReadBinary

Remarks

- If entering software standby soon after starting the RTC, use R_RTC_ReadBinary first to confirm that the values are correct.
- The oscillation accuracy of the sub-clock is affected when an on-chip debugger emulator is connected and the sub-clock drive setting is low.
- Before calling this function the count source must be enabled and stable. Refer R_CGC_Set and R_CGC_Control for count source and stabilization time configuration.
- This function is called to use RTC after setting option PDL_CGC_RTC_TO_BE_USED in R_CGC_Control at cold start.
- This function is not required when using 48-pin package.

Program example

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

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

void alarm_function(void){}

void func(void)
{
    /* Configure the count for an alarm at 0xXXXX4321 are matched */
    R_RTC_CreateBinary(
        PDL_NO_DATA,
        0x12345678,  /* The current count in 32 bits, binary display */
        PDL_NO_DATA, /* Periodic Interrupt Select */
        0x87654321,  /* The alarm count in 32 bits, binary display */
        0x0000ffff,  /* The alarm mask in 32 bits to enable alarm */
        alarm_function,
        15,
        PDL_NO_FUNC,
        PDL_NO_DATA
    );
}
```

3) R_RTC_Destroy

Synopsis

Shut down the Real-time clock.

Prototype

```
bool R_RTC_Destroy(
    void
);
```

Description

Stop the RTC counter and disable the sub-clock to the RTC.

Return value

True

Category

RTC

Reference

None.

Remarks

- This function is available for both calendar count mode and binary count mode.
- This function is not required when using 48-pin package.

Program example

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

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

void func(void)
{
    /* Shutdown the RTC */
    R_RTC_Destroy();
}
```

4) R_RTC_Control

Synopsis

Modify the Real-time clock operation.

Prototype

```
bool R_RTC_Control(
    uint32_t data1,     // Control selection
    uint16_t data2,     // Update selection
    uint32_t data3,     // Current time
    uint32_t data4,     // Current date
    uint32_t data5,     // Alarm Time
    uint32_t data6,     // Alarm Date
    uint16_t data7,     // Error Adjustment
    uint16_t data8      // Periodic configuration
);
```

Description (1/3)

Change clock settings and update the time or date in calendar count mode.

[data1]
Change the clock operation.
To set multiple options at the same time, use "|" to separate each value.
If no change is required, specify PDL_NO_DATA.

- 12 or 24 hour mode

PDL_RTC_24_HOUR_MODE or PDL_RTC_12_HOUR_MODE	Select 12 or 24 hour mode.

- Alarm control

PDL_RTC_ALARM_HOUR_DISABLE or PDL_RTC_ALARM_HOUR_ENABLE	All three can be controlled using: PDL_RTC_ALARM_TIME_DISABLE or PDL_RTC_ALARM_TIME_ENABLE
PDL_RTC_ALARM_MINUTE_DISABLE or PDL_RTC_ALARM_MINUTE_ENABLE	
PDL_RTC_ALARM_SECOND_DISABLE or PDL_RTC_ALARM_SECOND_ENABLE	
PDL_RTC_ALARM_YEAR_DISABLE or PDL_RTC_ALARM_YEAR_ENABLE	All four can be controlled using: PDL_RTC_ALARM_DATE_DISABLE or PDL_RTC_ALARM_DATE_ENABLE
PDL_RTC_ALARM_MONTH_DISABLE or PDL_RTC_ALARM_MONTH_ENABLE	
PDL_RTC_ALARM_DAY_DISABLE or PDL_RTC_ALARM_DAY_ENABLE	
PDL_RTC_ALARM_DOW_DISABLE or PDL_RTC_ALARM_DOW_ENABLE	

- Clock output control
  (RTC counting will be stopped temporarily during the writing of RTCOE bit)

PDL_RTC_OUTPUT_DISABLE or PDL_RTC_OUTPUT_ENABLE	Disable or enable the 1 Hz/64 Hz clock output on the RTCOUT pin.

- Clock RTCOUT output period Select
  (RTC counting will be stopped temporarily during the writing of RTCOS bit)

PDL_RTC_OUTPUT_RTCOS_1HZ or PDL_RTC_OUTPUT_RTCOS_64HZ	RTCOUT outputs 1-Hz RTCOUT outputs 64-Hz

- Clock control

PDL_RTC_CLOCK_STOP or PDL_RTC_CLOCK_START	Stop or re-start the clock.

- 30-second adjustment control

PDL_RTC_ADJUST_START	Start the 30-second adjustment process.

- Reset control

PDL_RTC_RESET_START	Start the reset process.

Description (2/3)

[data2]
Select the values to be changed.
To set multiple options at the same time, use "|" to separate each value.
If no change is required, specify PDL_NO_DATA.

- Select the time counters to be updated, using values supplied in parameter data3.

PDL_RTC_UPDATE_CURRENT_HOUR	All three can be selected using: PDL_RTC_UPDATE_CURRENT_TIME
PDL_RTC_UPDATE_CURRENT_MINUTE	
PDL_RTC_UPDATE_CURRENT_SECOND	

- Select the date counters to be updated, using values supplied in parameters data3 and data4.

PDL_RTC_UPDATE_CURRENT_YEAR	All four can be selected using: PDL_RTC_UPDATE_CURRENT_DATE. Parameter data3 is used for the day of the week.
PDL_RTC_UPDATE_CURRENT_MONTH	
PDL_RTC_UPDATE_CURRENT_DAY	
PDL_RTC_UPDATE_CURRENT_DOW	

- Select the alarm time counters to be updated, using values supplied in parameter data5.

PDL_RTC_UPDATE_ALARM_HOUR	All three can be selected using PDL_RTC_UPDATE_ALARM_TIME.
PDL_RTC_UPDATE_ALARM_MINUTE	
PDL_RTC_UPDATE_ALARM_SECOND	

- Select the alarm date counters to be updated, using values supplied in parameters data5 and data6.

PDL_RTC_UPDATE_ALARM_YEAR	All four can be selected using PDL_RTC_UPDATE_ALARM_DATE. Parameter data5 is used for the day of the week.
PDL_RTC_UPDATE_ALARM_MONTH	
PDL_RTC_UPDATE_ALARM_DAY	
PDL_RTC_UPDATE_ALARM_DOW	

[data3]
The new day of the week and time. Ignored if not selected above.
See R_RTC_Create for the format.

[data4]
The new year, month and day. Ignored if not selected above.
See R_RTC_Create for the format.

[data5]
The new alarm day of the week and time. Ignored if not selected above.
See R_RTC_Create for the format.

[data6]
The new alarm year, month and day. Ignored if not selected above.
See R_RTC_Create for the format.

[data7]
Configure the Error Adjustment options.
To set multiple options at the same time, use "|" to separate each value.
If no change is required, specify PDL_NO_DATA.

- Auto Error Adjustment

PDL_RTC_ERROR_AUTO_ADJUST_DISABLE or PDL_RTC_ERROR_AUTO_ADJUST_ENABLE	Enable or disable automatic error adjustment.

- Auto Error Adjustment Period

PDL_RTC_ERROR_AUTO_ADJUST_PERIOD_60S or PDL_RTC_ERROR_AUTO_ADJUST_PERIOD_10S	Select the automatic error adjustment period.

- Auto Error Adjustment Addition or subtraction selection

PDL_RTC_ERROR_ADJUST_PLUS or PDL_RTC_ERROR_ADJUST_MINUS	Select if the adjustment value will be added or subtracted from the count.

Description (3/3)

- Update the Error Adjustment value.

PDL_RTC_ERROR_UPDATE_ERROR_ADJUST_VALUE	Select to specify a new error adjustment value.

- Error Adjustment Value

Valid Range 0 to 3Fh	New automatic error adjustment value, ignored if not selected above.

[data8]
Configure the clock periodic interrupt.

- Periodic interrupt selection

PDL_RTC_PERIODIC_DISABLE or PDL_RTC_PERIODIC_256_HZ or PDL_RTC_PERIODIC_128_HZ or PDL_RTC_PERIODIC_64_HZ or PDL_RTC_PERIODIC_32_HZ or PDL_RTC_PERIODIC_16_HZ or PDL_RTC_PERIODIC_8_HZ or PDL_RTC_PERIODIC_4_HZ or PDL_RTC_PERIODIC_2_HZ or PDL_RTC_PERIODIC_1_HZ or PDL_RTC_PERIODIC_2S	The frequency or interval for periodic interrupt requests.

Return value

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

Category

Real-time clock

Reference

R_RTC_Create, R_RTC_Read

Remarks

- This function is not available in binary count mode.
- Refer to R_RTC_Create for the time and date formats
- If the current time or date values are updated, the clock is stopped during the update.
- If the day of week is updated using automatic calculation, the most recent year, month and date will be used.
- The range checking for either day value uses the most recent year and month values.
- If entering software standby mode soon after modifying the RTC values, use R_RTC_Read first to confirm that the values are correct.
- If the output of the RTCOUT pin is enabled or disabled, the clock is stopped during the update.
- This function is called after R_RTC_Create or R_RTC_CreateWarm.
- This function is not required when using 48-pin package.

Program example

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

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

void func(void)
{
    /* Disable the alarm calendar, and update the alarm time */
    R_RTC_Control(
        PDL_RTC_ALARM_DATE_DISABLE,
        PDL_RTC_UPDATE_ALARM_TIME,
        PDL_NO_DATA,
        PDL_NO_DATA,
        0x00105300,    /* Alarm at 10:53. */
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA
    );

    /* Change the day to the 23rd */
    R_RTC_Control(
        PDL_NO_DATA,
        PDL_RTC_UPDATE_CURRENT_DOW |
PDL_RTC_UPDATE_CURRENT_DAY,
        0xFF000000,
        0x00000023,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA
    );
}
```

5) R_RTC_ControlBinary

Synopsis

Modify the RTC operation in binary count mode.

Prototype

```
bool R_RTC_ControlBinary(
    uint32_t data1,     // Control selection
    uint32_t data2,     // Current count
    uint32_t data3,     // Alarm count
    uint32_t data4,     // Alarm mask
    uint16_t data5,     // Error Adjustment
    uint16_t data6      // Periodic configuration
);
```

Description (1/2)

Change RTC settings and update counter in binary count mode.

[data1]
Change the clock operation.
To set multiple options at the same time, use "|" to separate each value.
If no change is required, specify PDL_NO_DATA.

- Clock output control

PDL_RTC_OUTPUT_DISABLE or PDL_RTC_OUTPUT_ENABLE	Disable or enable the 1 Hz/64 Hz clock output on the RTCOUT pin.

- Clock RTCOUT output period Select
  (RTC counting will be stopped during writing RTCOS)

PDL_RTC_OUTPUT_RTCOS_1HZ or PDL_RTC_OUTPUT_RTCOS_64HZ	RTCOUT outputs 1-Hz RTCOUT outputs 64-Hz

- Clock control

PDL_RTC_CLOCK_STOP or PDL_RTC_CLOCK_START	Stop or re-start the clock.

- Reset control

PDL_RTC_RESET_START	Start the reset process.

[data2]
The new current count value. Count seconds in 32 bits, binary display.
If not required, specify PDL_NO_DATA.

[data3]
The new alarm count value. Setting the alarm register corresponding to 32-bit binary counter.
If not required, specify PDL_NO_DATA.

[data4]
The new alarm mask-value in 32 bits binary format.Setting the alarm enable corresponding to 32-bit binary alarm counter.
If not required, specify PDL_NO_DATA.

[data5]
Configure the Error Adjustment options.
To set multiple options at the same time, use "|" to separate each value.
If no change is required, specify PDL_NO_DATA.

- Auto Error Adjustment

PDL_RTC_ERROR_AUTO_ADJUST_DISABLE or PDL_RTC_ERROR_AUTO_ADJUST_ENABLE	Enable or disable automatic error adjustment.

- Auto Error Adjustment Period

PDL_RTC_ERROR_AUTO_ADJUST_PERIOD_32S or PDL_RTC_ERROR_AUTO_ADJUST_PERIOD_8S	Select the automatic error adjustment period.

- Auto Error Adjustment Addition or subtraction selection

PDL_RTC_ERROR_ADJUST_PLUS or PDL_RTC_ERROR_ADJUST_MINUS	Select if the adjustment value will be added or subtracted from the count.

Description (2/2)

- Update the Error Adjustment value.

PDL_RTC_ERROR_UPDATE_ERROR_ADJUST_VALUE	Select to specify a new error adjustment value.

- Error Adjustment Value

Valid Range 0 to 3Fh	New automatic error adjustment value, ignored if not selected above.

[data6]
Configure the clock periodic interrupt.

- Periodic interrupt selection

PDL_RTC_PERIODIC_DISABLE or PDL_RTC_PERIODIC_256_HZ or PDL_RTC_PERIODIC_128_HZ or PDL_RTC_PERIODIC_64_HZ or PDL_RTC_PERIODIC_32_HZ or PDL_RTC_PERIODIC_16_HZ or PDL_RTC_PERIODIC_8_HZ or PDL_RTC_PERIODIC_4_HZ or PDL_RTC_PERIODIC_2_HZ or PDL_RTC_PERIODIC_1_HZ or PDL_RTC_PERIODIC_2S	The frequency or interval for periodic interrupt requests.

Return value

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

Category

Real-time clock

Reference

R_RTC_CreateBinary, R_RTC_ReadBinary

Remarks

- This function is not available in calendar count mode.
- If the current count values are updated, the RTC is stopped during the update.
- If entering software standby mode soon after modifying the RTC values, use R_RTC_ReadBinary first to confirm that the values are correct.
- If the output of the RTCOUT pin is enabled or disabled, the clock is stopped during the update.
- This function is called after R_RTC_CreateBinary or R_RTC_CreateWarm.
- This function is not required when using 48-pin package.

Program example

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

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

void func(void)
{
    /* Disable the alarm, and update the alarm count */
    R_RTC_ControlBinary(
        PDL_NO_DATA,
        PDL_NO_DATA,
        0x23456789,    /* update the alarm count */
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA
    );
}
```

6) R_RTC_Read

Synopsis

Read the Real-time clock status flags and counters.

Prototype

```
bool R_RTC_Read(
    uint8_t data1,      // Specify what to read
    uint8_t * data2,    // A pointer to the flags storage location
    uint32_t * data3,   // A pointer to the data storage location
    uint32_t * data4    // A pointer to the data storage location
);
```

Description

Read the Clock counters registers and status flags in calendar count mode.

[data1]
Specify what to read.

PDL_RTC_READ_CURRENT or PDL_RTC_READ_ALARM	Specify which time to read.

[data2]
The format of data2 is dependent upon data1.

Format if data1 = PDL_RTC_READ_CURRENT

The clock status shall be stored in the following format.
Specify PDL_NO_PTR if the flags are not to be read.

b7	b6	b5	b4
Mode	Interrupt requests		
0: 12 hour 1: 24 hour	Carry	Periodic	Alarm
	0: Idle 1: Occurred		

b3	b2	b1	b0
0	Status		
	30-second adjustment	Reset	Clock
	0: Normal operation 1: Adjustment in progress	0: Normal operation 1: Reset in progress	0: Stopped 1: Running

Format if data1 = PDL_RTC_READ_ALARM

The enable bits for the alarm shall be stored in the following format.
1 = enabled, meaning the unit is part of the alarm setting.
0 = disabled, meaning the unit is ignored.
Specify PDL_NO_PTR if the flags are not to be read.

b7	b6	b5	b4	b3	b2	b1	b0
0	Year	Month	Day	Day of week	Hours	Minutes	Seconds

[data3]
The day of the week and time. Specify PDL_NO_PTR if it is not required.
See R_RTC_Create for the format.

[data4]
The year, month and day.Specify PDL_NO_PTR if it is not required.
See R_RTC_Create for the format.

Return value

True if all parameters are valid; otherwise false.

Category

Real-time clock

Reference

R_RTC_Create

Remarks

- This function is not available in Binary count mode
- Call R_RTC_Create or R_RTC_CreateWarm first before using this function.
- If an interrupt request flag is set to 1, it shall be automatically cleared to 0 by this function.
- Refer to R_RTC_Create for the time and date formats.
- If the Carry flag is read as 1, the current time and date were updated during the read process and should be re-read.
- The year returned will be in the range 0 to 99. The hundreds and thousands units are not stored.
- This function is not required when using 48-pin package.

Program example

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

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

uint8_t Flags;
uint32_t CurrentTime;

void func(void)
{
    /* Read the current time and flags */
    R_RTC_Read(
        PDL_RTC_READ_CURRENT,
        &Flags,
        &CurrentTime,
        PDL_NO_PTR
    );
}
```

7) R_RTC_ReadBinary

Synopsis

Read the Real-time clock status flags and counters.

Prototype

```
bool R_RTC_ReadBinary(
    uint8_t * data1,     // A pointer to the flags storage location
    uint32_t * data2,    // A pointer to the data storage location
    uint32_t * data3     // A pointer to the data storage location
    uint32_t * data4     // A pointer to the data storage location
);
```

Description

Read the Clock counters registers and status flags in binary count mode.

[data1]
The clock status shall be stored in the following format.
Specify PDL_NO_PTR if the flags are not to be read.

b7	b6	b5	b4
0	Interrupt requests		
	Carry	Periodic	Alarm
	0: Idle 1: Occurred		

b3 ~ b2	b1	b0
0	Status	
	Reset	Clock
	0: Normal operation 1: Reset in progress	0: Stopped 1: Running

[data2]
The current count in 32 bits, binary display. Specify PDL_NO_PTR if it is not required.

[data3]
The alarm count in 32 bits, binary display. Specify PDL_NO_PTR if it is not required.

[data4]
The alarm mask-data in 32 bits, binary display. Specify PDL_NO_PTR if it is not required.

Return value

True if all parameters are valid; otherwise false

Category

Real-time clock

Reference

R_RTC_ReadBinary

Remarks

- This function is not available in calendar count mode
- Call R_RTC_CreateBinary or R_RTC_CreateWarm first before using this function.
- If an interrupt request flag is set to 1, it shall be automatically cleared to 0 by this function.
- If the Carry flag is read as 1, the current counts were updated during the read process and should be re-read.
- This function is not required when using 48-pin package.

Program example

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

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

uint8_t Flags;
uint32_t Count;
uint32_t Alarm;

void func(void)
{
    /* Read count, alarm and flags */
    R_RTC_ReadBinary(
        &Flags,
        &Count,
        &Alarm,
        PDL_NO_PTR
    );
}
```

8) R_RTC_CreateWarm

Synopsis

Reconfigure RTC interrupt setting at warm start up in calendar count or binary count mode.

Prototype

```
bool R_RTC_CreateWarm(
    void * func1,       // Callback function
    uint8_t data1,      // Interrupt priority level
    void * func2,       // Callback function
    uint8_t data2       // Interrupt priority level
);
```

Description

Reconfigure RTC interrupt setting at warm start up.

[func1]
The function to be called when an alarm occurs. Specify PDL_NO_FUNC if not required.

[data1]
The alarm 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 func1.

[func2]
The function to be called at the periodic interval. Specify PDL_NO_FUNC if not required.

[data2]
The periodic 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 func2.

Return value

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

Category

Real-time clock

Reference

R_CGC_Set, R_CGC_Control

Remarks

- The function is called only at warm start up.
- This function is not required when using 48-pin package.

Program example

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

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

void alarm_function(void){}

void func(void)
{
    /* Reconfigure RTC intterupt. */
    R_RTC_CreateWarm(
         alarm_function,
         15,
         PDL_NO_FUNC,
         PDL_NO_DATA
    );
}
```

4.2.19. Independent Watchdog Timer

1) R_IWDT_Set

Synopsis

Configure the Independent Watchdog operation.

Prototype

```
bool R_IWDT_Set(
    uint32_t data    // Configuration selection
);
```

Description

Select the operation of the Independent Watchdog timer and start it.

[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	Clock division ratio selection The IWDTCLK clock ÷ 1, 16, 32, 64, 128 or 256.

- Time out control

PDL_IWDT_TIMEOUT_NMI or **PDL_IWDT_TIMEOUT_RESET**	If the IWDT times out, select if a Reset or an NMI Interrupt will be generated.

- Window Start Position

PDL_IWDT_WIN_START_25 or PDL_IWDT_WIN_START_50 or PDL_IWDT_WIN_START_75 or **PDL_IWDT_WIN_START_100**	The window start position specified as a percentage of the down counter. 0% is when the down-counter would underflow. Selecting 100% is equivalent to no window start position.

- Window End Position

PDL_IWDT_WIN_END_0 or PDL_IWDT_WIN_END_25 or PDL_IWDT_WIN_END_50 or PDL_IWDT_WIN_END_75	The window end position specified as a percentage of the down counter. 0% is when the down-counter would underflow. Hence specifying 0% is equivalent to no window end position.

- Sleep Mode Count Stop

PDL_IWDT_STOP_DISABLE or **PDL_IWDT_STOP_ENABLE**	Enable or disable Count stop mode. If the Count Stop mode is enabled the IWDT counter is stopped at a transition to sleep mode, software standby mode, or all-module clock stop mode.

Return value

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

Category

Independent Watchdog Timer

Reference

R_MCU_OFS, R_CGC_Set, R_CGC_Control, R_INTC_CreateExtInterrupt

Remarks

- If using the Initial Setting Memory (using R_MCU_OFS) to enable the IWDT from reset, this function will have no affect and can be omitted.
- The IWDTCLK must be enabled using R_CGC_Set or R_CGC_Control.
- If configuring to use a NMI handler then R_INTC_CreateExtInterrupt must be used to enable the NMI for IWDT.
- The IWDT counter frequency must not be greater than the PCLB / 4. Set the IWDTCLK division ratio accordingly. This function will return false if this condition is detected.

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

Synopsis

Control the Independent Watchdog operation.

Prototype

```
bool R_IWDT_Control(
    uint8_t data    // Control selection
);
```

Description

Modify the operation of the Independent Watchdog timer.

[data]
Control the timer.

- Counter start / refresh

PDL_IWDT_REFRESH	Start or refresh the counter by re-loading the timeout value.

Return value

True if the parameter is valid; otherwise false.

Category

Independent Watchdog Timer

Reference

R_IWDT_Set

Remarks

- R_IWDT_Set must be used first to configure the timer unless using Initial Setting Memory (using R_MCU_OFS) to enable the IWDT from reset.

Program example

```
/* 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
    );
}
```

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 and current counter value.

[data]
The timer status shall be stored in the following format.

b15	b14	b13 – b0
Refresh Error	Underflow	Down Counter Value
0: No refresh error 1: Refresh error	0: No underflow 1: Underflow	

Return value

True.

Category

Independent Watchdog Timer

Reference

None.

Remarks

- If the Underflow flag is set to 1, it shall be automatically cleared to 0 by this function.
- If the Refresh 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
    );
}
```

4.2.20. Serial Communication Interface

1) R_SCI_Set

Synopsis

Configure the SCI pin selection for SCI channels where there is a choice of SCI pins.

Prototype

```
bool R_SCI_Set(
    uint8_t data1,      // Channel selection
    uint16_t data2      // Pin configuration
);
```

Description (1/2)

Configure I/O pins. All pins used must be specified. There is no default option.

[data1]
The channel number n (where n = 1, 5, 6, 9 or 12).
The 48 pin package does not support SCI channel 9.

[data2]
Configure the global options. Use “|” to separate each selection.

- Valid when n = 1

PDL_SCI_PIN_SCI1_RXD1_P15 or PDL_SCI_PIN_SCI1_RXD1_P30	SCI1	RXD1
PDL_SCI_PIN_SCI1_SMISO1_P15 or PDL_SCI_PIN_SCI1_SMISO1_P30		SMISO1
PDL_SCI_PIN_SCI1_SSCL1_P15 or PDL_SCI_PIN_SCI1_SSCL1_P30		SSCL1
PDL_SCI_PIN_SCI1_TXD1_P16 or PDL_SCI_PIN_SCI1_TXD1_P26		TXD1
PDL_SCI_PIN_SCI1_SMOSI1_P16 or PDL_SCI_PIN_SCI1_SMOSI1_P26		SMOSI1
PDL_SCI_PIN_SCI1_SSDA1_P16 or PDL_SCI_PIN_SCI1_SSDA1_P26		SSDA1
PDL_SCI_PIN_SCI1_SCK1_P17 or PDL_SCI_PIN_SCI1_SCK1_P27		SCK1
PDL_SCI_PIN_SCI1_CTS1_P14 or PDL_SCI_PIN_SCI1_CTS1_P31		CTS1
PDL_SCI_PIN_SCI1_RTS1_P14 or PDL_SCI_PIN_SCI1_RTS1_P31		RTS1
PDL_SCI_PIN_SCI1_SS1_P14 or PDL_SCI_PIN_SCI1_SS1_P31		SS1

- Valid when n = 5

PDL_SCI_PIN_SCI5_RXD5_PA2 or PDL_SCI_PIN_SCI5_RXD5_PA3 or PDL_SCI_PIN_SCI5_RXD5_PC2	SCI5	RXD5
PDL_SCI_PIN_SCI5_SMISO5_PA2 or PDL_SCI_PIN_SCI5_SMISO5_PA3 or PDL_SCI_PIN_SCI5_SMISO5_PC2		SMISO5
PDL_SCI_PIN_SCI5_SSCL5_PA2 or PDL_SCI_PIN_SCI5_SSCL5_PA3 or PDL_SCI_PIN_SCI5_SSCL5_PC2		SSCL5
PDL_SCI_PIN_SCI5_TXD5_PA4 or PDL_SCI_PIN_SCI5_TXD5_PC3		TXD5
PDL_SCI_PIN_SCI5_SMOSI5_PA4 or PDL_SCI_PIN_SCI5_SMOSI5_PC3		SMOSI5
PDL_SCI_PIN_SCI5_SSDA5_PA4 or PDL_SCI_PIN_SCI5_SSDA5_PC3		SSDA5
PDL_SCI_PIN_SCI5_SCK5_PA1 or PDL_SCI_PIN_SCI5_SCK5_PC1 or PDL_SCI_PIN_SCI5_SCK5_PC4		SCK5
PDL_SCI_PIN_SCI5_CTS5_PA6 or PDL_SCI_PIN_SCI5_CTS5_PC0		CTS5
PDL_SCI_PIN_SCI5_RTS5_PA6 or PDL_SCI_PIN_SCI5_RTS5_PC0		RTS5
PDL_SCI_PIN_SCI5_SS5_PA6 or PDL_SCI_PIN_SCI5_SS5_PC0		SS5

- Valid when n = 6

PDL_SCI_PIN_SCI6_RXD6_P33 or PDL_SCI_PIN_SCI6_RXD6_PB0	SCI6	RXD6
PDL_SCI_PIN_SCI6_SMISO6_P33 or PDL_SCI_PIN_SCI6_SMISO6_PB0		SMISO6
PDL_SCI_PIN_SCI6_SSCL6_P33 or PDL_SCI_PIN_SCI6_SSCL6_PB0		SSCL6
PDL_SCI_PIN_SCI6_TXD6_P32 or PDL_SCI_PIN_SCI6_TXD6_PB1		TXD6
PDL_SCI_PIN_SCI6_SMOSI6_P32 or PDL_SCI_PIN_SCI6_SMOSI6_PB1		SMOSI6
PDL_SCI_PIN_SCI6_SSDA6_P32 or PDL_SCI_PIN_SCI6_SSDA6_PB1		SSDA6
PDL_SCI_PIN_SCI6_SCK6_P34 or PDL_SCI_PIN_SCI6_SCK6_PB3		SCK6
PDL_SCI_PIN_SCI6_CTS6_PB2 or PDL_SCI_PIN_SCI6_CTS6_PJ3		CTS6
PDL_SCI_PIN_SCI6_RTS6_PB2 or PDL_SCI_PIN_SCI6_RTS6_PJ3		RTS6
PDL_SCI_PIN_SCI6_SS6_PB2 or PDL_SCI_PIN_SCI6_SS6_PJ3		SS6

Valid when n = 9

PDL_SCI_PIN_SCI9_RXD9_PB6	SCI9	RXD9
PDL_SCI_PIN_SCI9_SMISO9_PB6		SMISO9
PDL_SCI_PIN_SCI9_SSCL9_PB6		SSCL9
PDL_SCI_PIN_SCI9_TXD9_PB7		TXD9
PDL_SCI_PIN_SCI9_SMOSI9_PB7		SMOSI9
PDL_SCI_PIN_SCI9_SSDA9_PB7		SSDA9
PDL_SCI_PIN_SCI9_SCK9_PB5		SCK9
PDL_SCI_PIN_SCI9_CTS9_PB4		CTS9
PDL_SCI_PIN_SCI9_RTS9_PB4		RTS9
PDL_SCI_PIN_SCI9_SS9_PB4		SS9

Description (2/2)

- Valid when n = 12

PDL_SCI_PIN_SCI12_RXD12_PE2	SCI12	RXD12
PDL_SCI_PIN_SCI12_SMISO12_PE2		SMISO12
PDL_SCI_PIN_SCI12_SSCL12_PE2		SSCL12
PDL_SCI_PIN_SCI12_TXD12_PE1		TXD12
PDL_SCI_PIN_SCI12_SMOSI12_PE1		SMOSI12
PDL_SCI_PIN_SCI12_SSDA12_PE1		SSDA12
PDL_SCI_PIN_SCI12_SCK12_PE0		SCK12
PDL_SCI_PIN_SCI12_CTS12_PE3		CTS12
PDL_SCI_PIN_SCI12_RTS12_PE3		RTS12
PDL_SCI_PIN_SCI12_SS12_PE3		SS12

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 using SCI channel 1, 5, 6, 9 or 12 call this function to configure the relevant pins.
- Pins which are not used for the SCI functions may be omitted.
- This function configures each specified SCI pin. It also disables the alternative modes on those pins.
- Please refer to the “Multifunction Pin Controller (MPC)” section in the RX220 Hardware Manual for details of SCI pin selection.
- Not all pins are available on all device package sizes.

Program example

```
#include "r_pdl_sci.h"

void func(void)
{
    /* Configure RXD1 and TXD1 pins*/
    R_SCI_Set(
        1,
        PDL_SCI_PIN_SCI1_RXD1_P15 | \
        PDL_SCI_PIN_SCI1_TXD1_P16
    );
}
```

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 (1/4)

Set up the selected SCI channel.

[data1]
Select channel SCIn (where n = 1, 5, 6, 9 or 12).

[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 (includes SPI and IIC), Smart Card Interface or Multi-Processor Asynchronous operation.

- Transmit / Receive connections (Not applicable in IIC Mode, option will be ignored.)

PDL_SCI_TX_CONNECTED or PDL_SCI_TX_DISCONNECTED	The TXDn output is required / not required.
PDL_SCI_RX_CONNECTED or PDL_SCI_RX_DISCONNECTED	The RXDn input is required / not required.

- Data transfer format (Not applicable in IIC Mode, option will be ignored.)

PDL_SCI_LSB_FIRST or PDL_SCI_MSB_FIRST	Select least or most significant bit first.

Options which are available in Asynchronous mode or Multi-Processor Asynchronous mode

- Noise Filter

PDL_SCI_RX_FILTER_DISABLE or PDL_SCI_RX_FILTER_ENABLE	Enable or disable the Digital Noise Filter on the RXDn pin.

- Hardware Flow Control

PDL_SCI_HW_FLOW_NONE or PDL_SCI_HW_FLOW_CTS or PDL_SCI_HW_FLOW_RTS	Select the Hardware Flow Control Option. Note: CTS and RTS functions cannot both be used as they share the same pin.

- Data clock source selection

PDL_SCI_CLK_INT_IO or	Select the on-chip baud rate generator.	SCKn pin: available as an I/O pin.
PDL_SCI_CLK_INT_OUT or		SCKn pin: SCI bit clock output.
PDL_SCI_CLK_EXT or	Input a clock of 8 or 16 times the desired bit rate to the SCKn pin. See parameter data3 for the multiplier selection.	
PDL_SCI_CLK_TMR	For SCI5, select Timer output TMO0, TMO1. For SCI6, select Timer output TMO2, TMO3. For SCI12, select Timer output TMO0, TM01. The SCKn pin is set to high-impedance.	

- Data length

PDL_SCI_8_BIT_LENGTH or PDL_SCI_7_BIT_LENGTH	8- or 7-bit data length.

Description (2/4)

- Parity mode

PDL_SCI_PARITY_NONE or PDL_SCI_PARITY_EVEN or PDL_SCI_PARITY_ODD	No parity bit, even parity bit or odd parity bit. Note: Do not set parity bit for Multi-Processor Asynchronous mode.

- Start bit edge detection

PDL_SCI_START_DETECT_LOW_LEVEL or PDL_SCI_START_DETECT_FALLING_EDGE	The low level or a falling edge on the RXDn pin is detected as the start bit.

- Stop bit length

PDL_SCI_STOP_1 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.

Options which are available in all Clock Synchronous modes (including IIC and SPI)

- SPI mode selection

PDL_SCI_SPI_MODE	SPI Mode selected: Use the R_SCI_SPI_Transfer function, not R_SCI_Send or R_SCI_Receive.

- IIC mode selection

PDL_SCI_IIC_MODE	IIC Mode selected: Use the functions R_SCI_IIC_Read and R_SCI_IIC_Write, not R_SCI_Send or R_SCI_Receive.

Options which are available in Clock Synchronous and SPI mode

- Data clock source selection

PDL_SCI_CLK_INT_OUT or	Select the On-chip baud rate generator. The SCKn pin outputs the bit clock. (In SPI Mode this is Master mode.)
PDL_SCI_CLK_EXT	Input the clock to the SCKn pin. (In SPI Mode this is Slave mode.)

- SPI Clock Polarity Inversion.

PDL_SCI_CLOCK_POLARITY_INVERTED	The SCK clock is inverted.

- SPI Clock Phase Delay

PDL_SCI_CLOCK_PHASE_DELAYED	The SCK clock is delayed.

Options which are available in Clock Synchronous mode (Not SPI or IIC)

- Hardware Flow Control

PDL_SCI_HW_FLOW_NONE or PDL_SCI_HW_FLOW_CTS or PDL_SCI_HW_FLOW_RTS	Select the Hardware Flow Control Option. Notes: • CTS can only be selected if using an internal clock source for SCLK. • RTS can only be selected if using external clock source for SCLK.

Options which are available in SPI mode

- SPI SS Pin

PDL_SCI_SPI_SS_DISABLE or	The SS pin is not used (Single master or single slave environment).
PDL_SCI_SPI_SS_ENABLE	The SS pin is used. Note: This option is not available if using SPI Master mode, if selected the function will return false.

- Data inversion

PDL_SCI_INVERSION_OFF or PDL_SCI_INVERSION_ON	Control data inversion (transmission and reception).

Description (3/4)

Options which are available in IIC mode

- Noise Filter Clock Select

PDL_SCI_IIC_FILTER_DISABLED or	The noise filter is disabled.
PDL_SCI_IIC_FILTER_CLOCK_DIV1 or PDL_SCI_IIC_FILTER_CLOCK_DIV2 or PDL_SCI_IIC_FILTER_CLOCK_DIV4 or PDL_SCI_IIC_FILTER_CLOCK_DIV8	The clock signal ÷ 1, 2, 4 or 8 is used with the noise filter.

- SSDA Delay Output Select (Delay on SDA Pin relative to SCL pin.)

PDL_SCI_IIC_DELAY_SDA_0_1 or	0 to 1 cycle delay
PDL_SCI_IIC_DELAY_SDA_1_2 or	1 to 2 cycle delay
PDL_SCI_IIC_DELAY_SDA_2_3 or	2 to 3 cycle delay
… (sequence continues)	…
PDL_SCI_IIC_DELAY_SDA_29_30 or	29 to 30 cycle delay
PDL_SCI_IIC_DELAY_SDA_30_31	30 to 31 cycle delay

Options which are available in Smart Card Interface mode

- Data inversion

PDL_SCI_INVERSION_OFF or PDL_SCI_INVERSION_ON	Control data inversion (transmission and reception).

- Base clock pulse cycle count

PDL_SCI_BCP_32 or PDL_SCI_BCP_64 or **PDL_SCI_BCP_93** or PDL_SCI_BCP_128 or PDL_SCI_BCP_186 or PDL_SCI_BCP_256 or PDL_SCI_BCP_372 or PDL_SCI_BCP_512	The number of base clock cycles in a 1-bit data transfer period.

- Parity selection

PDL_SCI_PARITY_EVEN or PDL_SCI_PARITY_ODD	Select even or odd parity bit.

- Block transfer mode selection

PDL_SCI_BLOCK_MODE_OFF or PDL_SCI_BLOCK_MODE_ON	Control Block transfer mode.

- GSM mode selection

PDL_SCI_GSM_MODE_OFF or PDL_SCI_GSM_MODE_ON	Control GSM mode.

- SCKn pin output control
  Note how the default option changes depending upon the mode. In Normal Mode the default is an I/O Pin. In GSM Mode the default is Fixed Low.

	Normal mode	GSM mode
PDL_SCI_SCK_OUTPUT_OFF or	I/O pin	Not applicable
PDL_SCI_SCK_OUTPUT_LOW or	Not applicable.	Fixed low.
PDL_SCI_SCK_OUTPUT_ON or	Outputs the bit clock.	
PDL_SCI_SCK_OUTPUT_HIGH	Not applicable	Fixed high.

Description (4/4)

[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 expected transfer bit rate.

- ABCS selection (required for asynchronous mode)

PDL_SCI_CYCLE_BIT_16 or PDL_SCI_CYCLE_BIT_8	Select 16 or 8 base clock cycles for one bit period.

- 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 PCLKB ÷ 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 may be zero if the following functions will not be used with a callback function:
R_SCI_Send, R_SCI_Receive, R_SCI_SPI_Transfer, R_SCI_IIC_Write and R_SCI_IIC_Read.

Return value

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

Category

SCI

Reference

R_CGC_Set, R_SCI_Set, R_SCI_Send, R_SCI_Receive, R_SCI_Control

Remarks

- Function R_CGC_Set must be called (with the current clock source selected) before using this function.
- Function R_SCI_Set must be called before any use of this function if using SCI channels 1, 5 or 6.
- This function configures each SCI pin that is required for operation. It also disables the alternative modes on those pins.
- SCI5 cannot be configured for IIC mode if the ELC event PDL_ELC_LINK_EVENT_SCI5_RECEIVE_DATA_FULL is being used. This function will return false if this condition is detected.
- In Async and Async MP modes the Tx pin is initially set to the Mark state. The R_SCI_Control function can subsequently be used to set the Space state.
- SPI Multi-Master mode is not supported. Hence, in SPI Master mode the SS pin cannot be enabled.
- If the option of using a delayed clock phase is selected in synchronous mode then a delay is required following the final receive interrupt before the operation can be completed. This delay is implemented as a software loop in the SCI RXI interrupt routine. See source file Interrupt_SCI.c for details.
- PDL_SCI_SPI_MODE cannot be selected for SCI Channel 12.
- The range of achievable bit rates (bps) is listed below.

Mode	Data clock source	Limit	f_{PCLKB}			
			32 MHz	12.5 MHz	12 MHz	8 MHz
Asynchronous	Internal	Minimum	62	24	23	16
		Maximum	2,000,000	781,250	750,000	500,000
	External		1,000,000	390,625	375,000	250,000
Synchronous	Internal	Minimum	489	191	184	123
		Maximum	4,000,000	1,562,500	1,500,000	1,000,000
	External		5,333,333	2,083,333	2,000,000	1,333,333
Smart card	Internal	Minimum	2	1	1	1
		Maximum	500,000	195,312	187,500	125,000

Program example

```
/* RPDL definitions */
#include "r_pdl_sci.h”

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

void func(void)
{
    /* Configure SCI1 for asynchronous, 8N1, 38400 baud */
    R_SCI_Create(
        1,
        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 = 1, 5, 6, 9 or 12).

Return value

True if all parameters are valid; otherwise false.

Category

SCI

Reference

None.

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
    uint16_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 = 1, 5, 6, 9 or 12).

[data2]
Control options.
The default options are shown in **bold**. Specify PDL_NO_DATA to use the defaults.

- DMAC / DTC trigger control.

PDL_SCI_DMAC_DTC_TRIGGER_DISABLE or PDL_SCI_DMAC_TRIGGER_ENABLE or PDL_SCI_DTC_TRIGGER_ENABLE	Disable or enable activation of the DMAC or DTC when a data byte is transmitted.

- ID transmission control (valid only in Multi-processor mode).

PDL_SCI_MP_ID_CYCLE	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 DMAC or 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 DMAC or 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.
DMAC	Either the function to be called when each byte is sent, or PDL_NO_FUNC if the callback function specified in R_DMAC_Create will be used.
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_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 5.18.7.
- For ID cycle, the DMAC / DTC trigger control and the callback function will be ignored.
- Do not use this function in SPI mode; use R_SCI_SPI_Transfer.
- Do not use this function in IIC mode; use R_SCI_IIC_Write.
- When using interrupts to manage the transfer, if the channel is operating in synchronous mode, transmit only and with an external clock, the TXD pin may need to be held active for longer (up to half a bit period) to avoid violating the data hold time for the receiving device. If a delay is required, the user should refer to the comments in the Transmit End interrupt processing routines (in the file Interrupt_SCI.c in the i_src folder) and implement the delay in a way that is suitable for their application.
- If using the DMAC or DTC this module does not know when the transmission has ended. Therefore when it has completed the user must call the R_SCI_Control function with option PDL_SCI_STOP_TX to manually disable the transmission.
- If a callback function is specified and the interrupt priority level is zero this function will return false.

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 1 */
    R_SCI_Send(
        1,
        PDL_NO_DATA,
        "Renesas RX",
        0,
        PDL_NO_FUNC
    );

    /* Send 50 bytes of binary data on channel 1 */
    R_SCI_Send(
        1,
        PDL_NO_DATA,
        data_store,
        50,
        PDL_NO_FUNC
    );

    /* Send the ID byte (0x0A, shifted into the upper byte) */
    R_SCI_Send(
        1,
        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_tdata1,       // Channel selection
    uint16_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 = 1, 5, 6, 9 or 12).

[data2]
Control options.
The default options are shown in **bold**. Specify PDL_NO_DATA to use the defaults.

- DMAC / DTC trigger control

PDL_SCI_DMAC_DTC_TRIGGER_DISABLE or PDL_SCI_DMAC_TRIGGER_ENABLE or PDL_SCI_DTC_TRIGGER_ENABLE	Disable or enable activation of the DMAC or DTC when a data byte is received.

- Continuous receive mode (valid only in asychronuous mode)

PDL_SCI_RX_CONTINUOUS_DISABLE or PDL_SCI_RX_CONTINUOUS_ENABLE	Disable or enable continuous receive when an interrupt is used as the receive method.

- 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 DMAC or 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 DMAC or 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.
DMAC	Either the function to be called when each byte is received, or PDL_NO_FUNC if the callback function specified in R_DMAC_Create will be used.
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_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 5.18.6.
- For the ID cycle, the DMAC / DTC trigger control will be ignored.
- In synchronous mode, if both the Tx Data and the Rx Data pins have been enabled (when R_SCI_Create was called), then a reception must be performed in conjunction with a corresponding transmission.
  This is achieved by calling R_SCI_Receive (in non-polling mode) and then R_SCI_Send.
  Please refer to the usage example in Section5.18.5.
- Do not use this function in SPI mode; use R_SCI_SPI_Transfer.
- Do not use this function in IIC mode; use R_SCI_IIC_Read.
- If using the DMAC or DTC this module does not know when the reception has ended. Therefore when it has completed the user must call the R_SCI_Control function with option PDL_SCI_STOP_RX to manually disable the reception.
- If a callback function is specified and the interrupt priority level is zero this function will return false.
- If PDL_SCI_RX_CONTINUOUS_ENABLE is selected, when next group of data is received after callback frunction, the data will be stored to the top of receive buffer (data3).

Program example

```
/* PDL functions */
#include "r_pdl_sci.h"

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

volatile uint8_t SCI1ReceiveBuffer[10];

/* SCI channel 1 receive data handler */
void SCI1RxFunc(void){}

/* SCI channel 1 error handler */
void SCI1ErrFunc(void){}

void func( void )
{
    uint8_t temp;

    /* Wait for 1 character to be received on channel 6 */
    R_SCI_Receive(
        6,
        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,
        SCI1ReceiveBuffer,
        9,
        SCI1RxFunc,
        SCI1ErrFunc
    );
}
```

6) R_SCI_SPI_Transfer

Synopsis

Perform an SPI transfer on an SCI channel.

Prototype

```
bool R_SCI_SPI_Transfer(
    uint8_tdata1,       // Channel selection
    uint16_t data2,     // Channel configuration
    uint16_t data3      // Number of bytes to transfer
    uint8_t * data4,    // Data transmit buffer
    void * func1,       // Callback function, Transmit Done
    uint8_t * data5,    // Data receive buffer
    void * func2,       // Callback function, Receive Done
    void * func3        // Callback function, Error
);
```

Description (1/2)

Perform an SPI transfer. This may be sending, receiving or both sending and receiving data.

[data1]
Select channel SCIn (where n = 1, 5, 6, 9 or 12).

[data2]
Control options.
The default options are shown in **bold**. Specify PDL_NO_DATA to use the defaults.

- DMAC / DTC trigger control

PDL_SCI_SPI_TX_DMAC_DTC_TRIGGER_DISABLE or PDL_SCI_SPI_TX_DMAC_TRIGGER_ENABLE or PDL_SCI_SPI_TX_DTC_TRIGGER_ENABLE	Disable or enable activation of the DMAC or DTC when a data byte is transmitted.

- DMAC / DTC trigger control

PDL_SCI_SPI_RX_DMAC_DTC_TRIGGER_DISABLE or PDL_SCI_SPI_RX_DMAC_TRIGGER_ENABLE or PDL_SCI_SPI_RX_DTC_TRIGGER_ENABLE	Disable or enable activation of the DMAC or DTC when a data byte is received.

[data3]
The number of bytes that must be transferred (either transmitted, received or both) before the function completes or the callback function is called.
If the DMAC or DTC shall be used to handle the received data, specify PDL_NO_DATA.

[data4]
The start address of the storage area for the expected data.
Specify PDL_NO_PTR if not transmitting data or if no data shall be processed by this function e.g. if the DMAC or DTC shall be used to process the received data.

[func1]
Transmit callback. 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.
Interrupts	The function to be called when the last byte has been sent.
DMAC	Either the function to be called when each byte is sent, or PDL_NO_FUNC if the callback function specified in R_DMAC_Create will be used.
DTC	The function to be called at the interval specified in R_DTC_Create.

[data5]
The start address of the storage area for the expected data.
Specify PDL_NO_PTR if not receiving data or if no data shall be processed by this function e.g. if the DMAC or DTC shall be used to process the received data.

Description (2/2)

[func2]
Receive callback. 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.
DMAC	Either the function to be called when each byte is received, or PDL_NO_FUNC if the callback function specified in R_DMAC_Create will be used.
DTC	The function to be called at the interval specified in R_DTC_Create.

[func3]
The function to be called if a receive error occurs. Specify PDL_NO_FUNC to ignore errors.

Return value

In Polling Mode:
True if all parameters are valid and the operation completed OK; false if a parameter was out of range or an error was detected.
In Non-Polling mode:
True if all parameters are valid; false if a parameter was out of range.

Category

SCI

Reference

R_SCI_Control, R_SCI_GetStatus

Remarks

- The maximum number of characters to be received or transmitted is 65535.
- Wait until a transmission on the same channel is complete before calling this function.
- If no error callback function (func3) 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 SPI master mode the slave(s) SS pin must be asserted before calling this function. A general I/O pin can be used for this, see the I/O Port API.
- If using the DMAC or DTC this module does not know when the transfer has ended. Therefore when the transfer has completed the user must call the R_SCI_Control function with options PDL_SCI_STOP_TX / PDL_SCI_STOP_RX to manually disable the transmission / reception as appropriate.
- If a callback function is specified and the interrupt priority level is zero this function will return false.
- If using this function to perform a full duplex transfer then the transfer mode for transmit and receive can be set independently. If using the polling transfer mode for only one direction this function must not be called from an interrupt handler so that interrupts can still be serviced for the non-polling transfer direction.

Program example

```
/* PDL functions */
#include "r_pdl_sci.h"

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

volatile uint8_t SCI1RxBuffer[10];
const uint8_t SCI1TxBuffer[10] =
{‘1’,‘2’,‘3’,‘4’,‘5’,‘6’,‘7’,‘8’,‘9’,‘0’};

/* SCI channel 1 receive data handler */
void SCI1RxFunc(void){}

/* SCI channel 1 error handler */
void SCI1ErrFunc(void){}

void func( void )
{
    /* Wait while send 5 characters on channel 6 */
    R_SCI_SPI_Transfer (
        6,
        PDL_NO_DATA,
        5,
        "12345",
        PDL_NO_FUNC,
        PDL_NO_DATA,
        PDL_NO_FUNC,
        PDL_NO_FUNC
    );

    /* Start the transmission and reception of 9 characters on channel
1 */
    R_SCI_Receive(
        1,
        PDL_NO_DATA,
        SCI1RxBuffer,
        9,
        SCI1RxFunc,
        SCI1ErrFunc
    );
}
```

7) R_SCI_IIC_Write

Synopsis

Perform an IIC master write on an SCI channel.

Prototype

```
bool R_SCI_IIC_Write(
    uint8_tdata1,       // Channel selection
    uint16_t data2,     // Channel configuration
    uint16_t data3,     // Slave Address
    uint16_t data4,     // Number of bytes to transfer
    uint8_t * data5,    // Buffer
    void * func         // Callback function.
);
```

Description (1/2)

Perform an IIC master write.

[data1]
Select channel SCIn (where n = 1, 5, 6, 9 or 12).

[data2]
Control options.
The default options are shown in **bold**. Specify PDL_NO_DATA to use the defaults.

- DMAC / DTC trigger control

PDL_SCI_IIC_DMAC_DTC_TRIGGER_DISABLE or PDL_SCI_IIC_DMAC_TRIGGER_ENABLE or PDL_SCI_IIC_DTC_TRIGGER_ENABLE	Disable or enable activation of the DMAC or DTC for the data stage.

- Slave Address Size

PDL_SCI_IIC_7_BIT_SLAVE_ADDRESS or PDL_SCI_IIC_10_BIT_SLAVE_ADDRESS	Specify the slave address width.

- Repeated Start

PDL_SCI_IIC_RESTART	The transfer will start with a re-start rather than the default behaviour of a start condition.

- Stop Condition selection

PDL_SCI_IIC_NOSTOP	By default the transfer will end with a stop condition. Select this option to prevent the stop condition being generated.

[data3]
Slave address, either 7 or 10 bits, use the format as specified here:

b15 - b8	b7 - b1	b0
-	7-bit address	-

b15 - b11	b10 - b1	b0
-	10-bit address	-

[data4]
The number of data bytes that must be transferred before the function completes or the callback function is called.
If the DMAC or DTC shall be used to handle the received data, specify PDL_NO_DATA.

[data5]
The start address of the buffer that contains the data to be written.
Specify PDL_NO_PTR if not transmitting data or if no data shall be processed by this function e.g. if the DMAC or DTC shall be used to send the data.

Description (2/2)

[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 transferred or an error occurs.
Interrupts	The function to be called when the transfer has completed or an error detected.
DMAC	Either the function to be called when each byte is transferred, or PDL_NO_FUNC if the callback function specified in R_DMAC_Create will be used.
DTC	The function to be called at the interval specified in R_DTC_Create.

Return value

In Polling Mode:
True if all parameters are valid and the operation completed OK; false if a parameter was out of range or an error was detected.
In Non-Polling mode:
True if all parameters are valid; false if a parameter was out of range.

Category

SCI

Reference

R_DMAC_Create, R_DTC_Create, R_SCI_Control

Remarks

- The maximum number of characters to be transmitted is 65535.
- Wait until a transmission on the same channel is complete before calling this function.
- Callback functions are executed by the interrupt processing function. This means that no other interrupt can be processed until a callback function has completed.
- This function, unless configured not to, will by default automatically start a transfer by generating a Start condition and finish with a Stop condition. However, if using DMAC or DTC the Stop condition will not be generated automatically, so use the R_SCI_Control function to manually generate a stop.
- If a callback function is specified and the interrupt priority level is zero this function will return false.
- The SCI IIC module is always configured to use Reception and Transmission interrupts (IICINTM bit = 1) rather than ACK/NACK interrupts. This means that if using the DMAC or DTC to transmit then all data will be transmitted even if the slave device fails to ACK.

Program example

```
/* PDL functions */
#include "r_pdl_sci.h"

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

#define CHANNEL_SCI_IIC 9
#define SLAVE_ADDRESS     0xA0

/* Buffer for IIC data */
extern uint8_t IIC_Buffer[10];

void func( void )
{
    /* Wait while send 10 bytes */
    R_SCI_IIC_Write(
        CHANNEL_SCI_IIC,
        PDL_NO_DATA,
        SLAVE_ADDRESS,
        10,
        IIC_Buffer,
        PDL_NO_FUNC
    );
}
```

8) R_SCI_IIC_Read

Synopsis

Perform an IIC master read on an SCI channel.

Prototype

```
bool R_SCI_IIC_Read(
    uint8_tdata1,       // Channel selection
    uint16_t data2,     // Channel configuration
    uint16_t data3,     // Slave Address
    uint16_t data4,     // Number of bytes to transfer
    uint8_t * data5,    // Buffer
    void * func         // Callback function.
);
```

Description (1/2)

Perform an IIC master read.

[data1]
Select channel SCIn (where n = 1, 5, 6, 9 or 12).

[data2]
Control options.
The default options are shown in **bold**. Specify PDL_NO_DATA to use the defaults.

- DMAC / DTC trigger control

PDL_SCI_IIC_DMAC_DTC_TRIGGER_DISABLE or PDL_SCI_IIC_DMAC_TRIGGER_ENABLE or PDL_SCI_IIC_DTC_TRIGGER_ENABLE	Disable or enable activation of the DMAC or DTC for the data stage.

- Slave Address Size

PDL_SCI_IIC_7_BIT_SLAVE_ADDRESS or PDL_SCI_IIC_10_BIT_SLAVE_ADDRESS	Specify the slave address width.

- Repeated Start

PDL_SCI_IIC_RESTART	The transfer will start with a re-start rather than the default behaviour of a start condition.

- Stop Condition selection

PDL_SCI_IIC_NOSTOP	By default the transfer will end with a stop condition. Select this option to prevent the stop condition being generated.

[data3]
Slave address, either 7 or 10 bits, use the format as specified here:

b15 - b8	b7 - b1	b0
-	7-bit address	-

b15 - b11	b10 - b1	b0
-	10-bit address	-

[data4]
The number of data bytes that must be transferred before the function completes or the callback function is called.
If the DMAC or DTC shall be used to handle the received data, specify PDL_NO_DATA.

[data5]
The start address of the buffer that will receive the data.
Specify PDL_NO_PTR if no data shall be processed by this function e.g. if the DMAC or DTC shall be used to process the received data.

Description (2/2)

[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 transferred or an error occurs.
Interrupts	The function to be called when the transfer has completed or an error detected.
DMAC	Either the function to be called when each byte is transferred, or PDL_NO_FUNC if the callback function specified in R_DMAC_Create will be used.
DTC	The function to be called at the interval specified in R_DTC_Create.

Return value

In Polling Mode:
True if all parameters are valid and the operation completed OK; false if a parameter was out of range or an error was detected.
In Non-Polling mode:
True if all parameters are valid; false if a parameter was out of range.

Category

SCI

Reference

R_SCI_GetStatus, R_SCI_IIC_ReadLastByte, R_SCI_Control

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.
- Callback functions are executed by the interrupt processing function. This means that no other interrupt can be processed until a callback function has completed.
- This function, unless configured not to, will by default automatically start a transfer by generating a Start condition and finish with a Stop condition. However, if using DMAC or DTC the Stop condition will not be generated automatically, so use the R_SCI_IIC_ReadLastByte or R_SCI_Control function to manually generate a stop.
- The last byte of a master read will automatically be NACK'd. However, if using DMAC or DTC this will not happen. If a NACK is required then use the DMAC / DTC to read all the data except for the last byte and then use function R_SCI_IIC_ReadLastByte to read the last byte.
- If a callback function is specified and the interrupt priority level is zero this function will return false.

Program example

```
/* PDL functions */
#include "r_pdl_sci.h"

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

#define CHANNEL_SCI_IIC 9
#define SLAVE_ADDRESS     0xA0

/* Buffer for IIC data */
extern uint8_t IIC_Buffer[10];

void func( void )
{
    /* Wait while read 10 bytes */
    R_SCI_IIC_Read(
        CHANNEL_SCI_IIC,
        PDL_NO_DATA,
        SLAVE_ADDRESS,
        10,
        IIC_Buffer,
        PDL_NO_FUNC
    );
}
```

9) R_SCI_IIC_ReadLastByte

Synopsis

Read the last byte of an IIC read transfer.

Prototype

```
bool R_SCI_IIC_ReadLastByte (
    uint8_t data1,      // Channel selection
    uint8_t * data2     // Buffer to receive byte.
);
```

Description

If R_SCI_IIC_Read has been used to start an IIC read where the DMAC or DTC will read all the data except for the last byte this function can be used to read the last byte.
A NACK will then be generated, followed by a stop condition (unless the original transfer request asked for the stop condition to be omitted).

[data1]
Select channel SCIn (where n = 1, 5, 6, 9 or 12).

[data2]
The address of the buffer that will receive the byte.

Return value

True.

Category

SCI

Reference

R_SCI_IIC_Read

Remarks

None

Program example

```
/* PDL functions */
#include "r_pdl_sci.h"

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

#define CHANNEL_SCI_IIC 9

/* Buffer for IIC data */
extern uint8_t IIC_Buffer[10];

void func( void )
{
    /* Read the last byte of the IIC read operation */
    R_SCI_IIC_ReadLastByte(
        CHANNEL_SCI_IIC,
        &IIC_Buffer[9]
    );
}
```

10) R_SCI_Control

Synopsis

Control the SCI channel.

Prototype

```
bool R_SCI_Control(
    uint8_t data1,      // Channel selection
    uint16_t data2      // Channel control
);
```

Description

Control the SCI channel.

[data1]
Select channel SCIn (where n = 1, 5, 6, 9 or 12).

[data2] (Not IIC Mode)
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.
  (Only applicable in Async and Async Multi-Processor Modes.)

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).

[data2] (IIC Mode only)
Control the channel.

- Stop condition generation

PDL_SCI_IIC_STOP	A stop will be output on the bus.

- Clock Synchronisation

PDL_SCI_IIC_CLOCK_SYNC_DISABLE or PDL_SCI_IIC_CLOCK_SYNC_ENABLE	Disable or enable the IIC clock synchronisation. Note: Clock synchronisation is enabled by default as required for normal operation.

Return value

True if all parameters are valid; otherwise false.

Category

SCI

Reference

None

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 1 */
    R_SCI_Control(
        1,
        PDL_SCI_STOP_RX
    );
}
```

11) R_SCI_GetStatus

Synopsis

Check the status of an 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 = 1, 5, 6, 9 or 12).

[data2]
The status flags shall be stored in one of the following formats depending on the current mode: (Note: Some bits are Not Applicable (NA) in all modes – see descriptions.)

Asynchronous or Synchronous modes: (Not IIC Mode)

b7-b6	b5	b4	b3	b2	b1	b0
0	Reception error detection			Transmit status	0	RxD pin level (NA to SPI mode)
	Overrun	Framing (Async. Mode Only)	Parity (Async. Mode Only)			
	0: No error 1: Detected	0: No error 1: Detected	0: No error 1: Detected	0: Active 1: Idle		0: Low 1: High

Smart card mode:

b7 – b6	b5	b4	b3	b2	b1	b0
0	Error detection			Transmit status	0	RxD pin level
	Overrun	Error signal	Parity			
	0: No error 1: Detected	0: No error 1: Detected	0: No error 1: Detected	0: Active 1: Idle		0: Low 1: High

IIC Mode:

b7 – b1	b0
0	ACK / NACK flag This is updated every time an ACK or NACK is received.
	0: ACK received 1: NACK received

[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.
NOTE: If using DMAC or DTC specify PDL_NO_PTR as this information is not available.

[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.
NOTE: If using DMAC or DTC specify PDL_NO_PTR as this information is not available.

Return value

True if all parameters are valid and the operation completed; false if a parameter was out of range or the RX pin has not been selected by using the R_SCI_Set and/or R_SCI_Create functions.

Category

SCI

Reference

R_SCI_Receive, R_SCI_Set

Remarks

- The error flags are not modified by this function. They are cleared when a new reception process is started.
- For channels SCI1, SCI5 and SCI6, if the RxD pin to be used has not been specified using R_SCI_Set before calling this, then the RxD pin level will always be read as low.

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 1 */
    R_SCI_GetStatus(
        1,
        &StatusValue,
        PDL_NO_PTR,
        &TxChars,
        &RxChars
    );
}
```

4.2.21. I²C Bus Interface

1) R_IIC_Set

Synopsis

Configure the I²C pin selection.

Prototype

```
bool R_IIC_Set(
    uint8_t data        // Pin selection
);
```

Description

Set up the selected I²C channel.

[data]
Configure the I²C pins. Use "|" to separate each selection.

- SDA pin selection

PDL_IIC_PIN_SDA_P13 or PDL_IIC_PIN_SDA_P17	Select the pin for SDA.

- SCL pin selection

PDL_IIC_PIN_SCL_P12 or PDL_IIC_PIN_SCL_P16	Select the pin for SCL.

Return value

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

Category

I²C

Reference

R_IIC_Create

Remarks

- Before calling R_IIC_Create, call this function to configure the relevant pins.
- Some pin options are not available on smaller device packages.

Program example

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

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

void func(void)
{
    /* Configure the applicable IIC pins */
    R_IIC_Set(
        PDL_IIC_PIN_SDA_P13 | PDL_IIC_PIN_SCL_P12
    );
}
```

2) R_IIC_Create

Synopsis

I²C 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 I²C 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 between I²C Bus or SMBus mode.

- Internal reference clock

PDL_IIC_INT_PCLK_DIV_1 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 (derived from PCLKB), used inside the I²C module.

- Timeout detection control

PDL_IIC_TIMEOUT_DISABLE 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

PDL_IIC_TIMEOUT_LONG or PDL_IIC_TIMEOUT_SHORT	Select 16-bit (long) or 14-bit (short) mode.

- SDA output delay count

PDL_IIC_SDA_DELAY_0 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

PDL_IIC_SDA_DELAY_DIV_1 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_DISABLE or **PDL_IIC_NF_1** or PDL_IIC_NF_2 or PDL_IIC_NF_3 or PDL_IIC_NF_4	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

I²C

Reference

R_CGC_Set, R_IIC_Set

Remarks

- Function R_CGC_Set must be called (with the current clock source selected) before using this function.
- Function R_IIC_Set must be called before any use of this function.
- This function configures each I²C 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_{PCLKB} (MHz)					
f_{IRC}	50	48	12.5	12	32	8
f_{PCLKB} ÷ 1	781 kbps to 25.0 Mbps	750 kbps to 24.0 Mbps	195 kbps to 6.25 Mbps	187.5 kbps to 6.0 Mbps	500 kbps to 16.0 Mbps	125 kbps to 4.00 Mbps
f_{PCLKB} ÷ 2	391 kbps to 12.5 Mbps	375 kbps to 12.0 Mbps	97.7 kbps to 3.13 Mbps	93.75 kbps to 3.0 Mbps	250 kbps to 8.00 Mbps	62.5 kbps to 2.00 Mbps
f_{PCLKB} ÷ 4	195 kbps to 6.25 Mbps	187.5 kbps to 6.0 Mbps	48.8 kbps to 1.56 Mbps	46.875 kbps to 1.5 Mbps	125 kbps to 4.00 Mbps	31.3 kbps to 1.00 Mbps
f_{PCLKB} ÷ 8	97.7 kbps to 3.13 Mbps	93.75 kbps to 3.0 Mbps	24.4 kbps to 781 kbps	23.4 kbps to 750 kbps	62.5 kbps to 2.00 Mbps	15.6 kbps to 500 kbps
f_{PCLKB} ÷ 16	48.8 kbps to 1.56 Mbps	46.875 kbps to 1.5 Mbps	12.2 kbps to 391 kbps	11.71 kbps to 375 kbps	31.3 kbps to 1.00 Mbps	7.81 kbps to 250 kbps
f_{PCLKB} ÷ 32	24.4 kbps to 781 kbps	23.4 kbps to 750 kbps	6.10 kbps to 195 kbps	5.86 kbps to 187.5 kbps	15.6 kbps to 500 kbps	3.91 kbps to 125 kbps
f_{PCLKB} ÷ 64	12.2 kbps to 391 kbps	11.71 kbps to 375 kbps	3.05 kbps to 97.7 kbps	2.93 kbps to 93.75 kbps	7.81 kbps to 250 kbps	1.95 kbps to 62.5 kbps
f_{PCLKB} ÷ 128	6.10 kbps to 195 kbps	5.86 kbps to 187.5 kbps	1.53 kbps to 48.8 kbps	1.46 kbps to 46.875 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 I²C 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:

	f_{PCLKB} (MHz)					
IRC	50	48	12.5	12	32	8
PCLKB ÷ 1	658 kbps to 1 Mbps	635.6 kbps to 1 Mbps	175 kbps to 1 Mbps	168.5 kbps to 1 Mbps	446 kbps to 1 Mbps	116 kbps to 1 Mbps
PCLKB ÷ 2	316 kbps to 1 Mbps	306 kbps to 1 Mbps	86.7 kbps to 1 Mbps	83.6 kbps to 1 Mbps	217 kbps to 1 Mbps	57.8 kbps to 1 Mbps
PCLKB ÷ 4	175 kbps to 1 Mbps	168.5 kbps to 1 Mbps	45.9 kbps to 1 Mbps	44.2 kbps to 1 Mbps	116 kbps to 1 Mbps	30.0 kbps to 806 kbps
PCLKB ÷ 8	86.7 kbps to 1 Mbps	83.6 kbps to 1 Mbps	23.7 kbps to 658 kbps	22.7 kbps to 635.6 kbps	57.8 kbps to 1 Mbps	15.3 kbps to 446 kbps
PCLKB ÷ 16	45.9 kbps to 1 Mbps	44.2 kbps to 1 Mbps	12.0 kbps to 316 kbps	11.5 kbps to 306.1 kbps	30.0 kbps to 806 kbps	7.73 kbps to 217 kbps
PCLKB ÷ 32	23.7 kbps to 658 kbps	22.7 kbps to 635.6 kbps	6.06 kbps to 175 kbps	5.8 kbps to 168.5 kbps	15.3 kbps to 446 kbps	3.89 kbps to 116 kbps
PCLKB ÷ 64	12.0 kbps to 316 kbps	11.5 kbps to 306.1 kbps	3.04 kbps to 86.7 kbps	2.9 kbps to 83.6 kbps	7.73 kbps to 217 kbps	1.95 kbps to 57.8 kbps
PCLKB ÷ 128	6.06 kbps to 175 kbps	5.82 kbps to 168.5 kbps	1.52 kbps to 45.9 kbps	1.5 kbps to 44.2 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 IIC 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 IIC 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
    );
}
```

3) R_IIC_Destroy

Synopsis

Disable an I²C channel.

Prototype

```
bool R_IIC_Destroy(
    uint8_t data     // Channel selection
);
```

Description

Shut down the selected I²C module.

[data]
Select channel IICn (where n = 0).

Return value

True if the parameter is valid; otherwise false.

Category

I²C

Reference

None

Remarks

- The I²C 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
    );
}
```

4) R_IIC_MasterSend

Synopsis

Write data to a slave device.

Prototype

```
bool R_IIC_MasterSend(
    uint8_t data1,      // Channel selection
    uint16_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(1/2)

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

PDL_IIC_START_ENABLE or PDL_IIC_START_DISABLE	Choose whether or not to issue a Start or Repeated Start condition at the beginning 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.

- Stop condition control

PDL_IIC_STOP_ENABLE or PDL_IIC_STOP_DISABLE	Choose whether or not to issue a Stop condition at the end of the transfer.

- DMAC / DTC trigger control

PDL_IIC_DMAC_DTC_TRIGGER_DISABLE or PDL_IIC_DMAC_TRIGGER_ENABLE or PDL_IIC_DTC_TRIGGER_ENABLE	Disable or enable activation of the DMAC or 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 DMAC or DTC shall be used to transfer the data, specify PDL_NO_PTR.

[data5]
The number of bytes to be sent.
If the DMAC or 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.
DMAC	Either the function to be called when each byte is sent, or PDL_NO_FUNC if the callback function specified in R_DMAC_Create will be used.
DTC	The function to be called at the interval specified in R_DTC_Create. This function will in addition also be called when the last byte has been transmitted.

Description (2/2)

[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 and a normal transfer completed; otherwise false.

Category

I²C

Reference

R_IIC_GetStatus, R_IIC_GetStatus, R_DMAC_Create

Remarks

- If a callback function is specified, transmission interrupts are used.
  Please see the notes on callback function usage in §6.
- If the Start condition is enabled and the previous transfer did not issue a Stop condition, a Repeated Start condition shall be generated.
- If the Start condition is disabled, the slave address will not be transmitted.
- If no callback function is specified for transmission completion, this function will monitor the status flags to manage the data transmission. If the I²C channel's registers are modified directly by the user, this function may lock up.
- If false is returned, use R_IIC_GetStatus to check if an unexpected event on I²C bus was the cause of the failure. If the transfer has ended prematurely, use R_IIC_Control to issue a Stop condition.
- False will be returned if the DMAC channel has not been allocated using R_DMAC_Create.
- False will be returned if the bus is busy due to another master on the bus.
- When using the DMAC or DTC a stop condition will not be generated. Hence, if one is required it must be manually generated using R_IIC_Control when it has been detected that the last byte has been sent.

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
    );
}
```

5) R_IIC_MasterReceive

Synopsis

Read data from a slave device.

Prototype

```
bool R_IIC_MasterReceive(
    uint8_t data1,      // Channel selection
    uint16_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 I²C channel and store it.

[data1]
Select channel IICn (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.

- DMAC / DTC trigger control

PDL_IIC_DMAC_DTC_TRIGGER_DISABLE or PDL_IIC_DMAC_TRIGGER_ENABLE or PDL_IIC_DTC_TRIGGER_ENABLE	Disable or enable activation of the DMAC or 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 DMAC or 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 DMAC or 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.
DMAC	Either the function to be called when each byte is received, or PDL_NO_FUNC if the callback function specified in R_DMAC_Create will be used.
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 I²C

Reference R_IIC_MasterReceiveLast, R_IIC_GetStatus, R_DMAC_Create

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 I²C 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 DMAC or DTC is used, use R_IIC_MasterReceiveLast to complete the transfer.
- Use R_IIC_GetStatus to determine if the transfer was successful.
- False will be returned if the DMAC channel has not been allocated using R_DMAC_Create.
- False will be returned if the bus is busy due to another master on the bus.

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
    );
}
```

6) R_IIC_MasterReceiveLast

Synopsis

Complete a DMAC or 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).

[data2]
The storage location for the data byte.

Return value

True if all parameters are valid and the function completed; otherwise false.

Category

I²C

Reference

R_IIC_GetStatus

Remarks

- This function must only be used to terminate a Read process that has used the DMAC or 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 DMAC or 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]
    );
}
```

7) R_IIC_SlaveMonitor

Synopsis

Monitor the bus.

Prototype

```
bool R_IIC_SlaveMonitor(
    uint8_t data1,      // Channel selection
    uint16_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 I²C channel.

[data1]
Select channel IICn (where n = 0).

[data2]
Select the operation options.
The default setting is shown in **bold**. Specify PDL_NO_DATA to use the default.

- DMAC / DTC trigger control

PDL_IIC_RX_DMAC_DTC_TRIGGER_DISABLE or PDL_IIC_RX_DMAC_TRIGGER_ENABLE or PDL_IIC_RX_DTC_TRIGGER_ENABLE	Disable or enable activation of the DMAC or DTC when a byte is received.
PDL_IIC_TX_DMAC_DTC_TRIGGER_DISABLE or PDL_IIC_TX_DMAC_TRIGGER_ENABLE or PDL_IIC_TX_DTC_TRIGGER_ENABLE	Disable or enable activation of the DMAC or DTC for data transmission.

[data3]
The start address of the storage area for any received data.
If the DMAC or DTC shall be used to handle the received data, specify PDL_NO_PTR.

[data4]
The number of bytes in the storage area.
If the DMAC or 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. If not using the DMAC or DTC this function will continue until a Stop or Re-Start condition is detected or the master tries to read data from this slave. If using the DMAC or DTC the function will return after detecting a slave address match so that the DTC/DMAC can complete the transfer.
Interrupts	The function to be called when a Stop or Re-Start condition is detected or the master tries to read data from this slave.
DMAC or DTC	The function to be called when a Stop or Re-Start 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

I²C

Reference

R_IIC_GetStatus, R_IIC_SlaveSend, R_DMAC_Create

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 using polling mode. When the function returns use R_IIC_GetStatus to identify the activity that has occurred.
- Call this function for each transfer required even if the master has ended the previous transfer with a repeat start.
- If the DMAC or DTC is not being used to perform a slave transmission then if a slave transmission is required function R_IIC_SlaveSend must be called to send the data.
  Note: If R_IIC_GetStatus reports that the slave is in transmit mode then a slave transmission is required.
- If the master sends more data than is expected and the DMAC / DTC trigger is disabled, this function will issue a NACK to the master.
- If using the DMAC or DTC for transferring data then ensure they are configured correctly before calling this function.
- False will be returned if the DMAC channel has not been allocated using R_DMAC_Create.
- Normally bus activity for other slaves is ignored with no CPU involvement. However, in the specific case where a callback function is specified and the DTC or DMAC is specified for data transmission, then any stop condition on the bus will cause the callback function to be called before any data has been transfered. This function should then be called again to continue monitoring the bus.

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
    );
}
```

8) 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

I²C

Reference

R_IIC_SlaveMonitor

Remarks

- Use this function after using R_IIC_SlaveMonitor and detecting that a slave transmission is required.
- If a callback function was specified in the call to R_IIC_SlaveMonitor then this transfer shall be completed using interrupts and the callback function shall be called when the transfer ends.
- If a callback function was not specified in the call to R_IIC_SlaveMonitor then this function will not return until the transfer has ended.
- If the master requires more data than is supplied this function shall loop back to the start of the data.

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
    );
}
```

9) R_IIC_Control

Synopsis

I²C channel control.

Prototype

```
bool R_IIC_Control(
    uint8_t data1,    // Channel selection
    uint8_t data2     // Control options
);
```

Description

Modify the operation of the selected I²C channel.

[data1]
Select channel IICn (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 I²C module (the settings are preserved).

Return value

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

Category

I²C

Reference

None

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
    );
}
```

10) R_IIC_GetStatus

Synopsis

Read the status for an I²C 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 I²C channel.

[data1]
Select channel IICn (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

I²C

Reference

R_DTC_GetStatus, R_DMAC_GetStatus.

Remarks

- The flags are not modified by this function. The event detection flags are cleared as required by the driver for correct operation. The transfer count values are cleared when a new transfer is started.
- If using the DTC or DMAC to transfer data the transfer count values will not be valid. The R_DTC_GetStatus or R_DMAC_GetStatus functions can be used to calculate the transfer count. Note: If the DTC / DMAC transfer does not fully complete then the count reported by the DTC / DMAC for a slave transmission will be one greater than the actual number of bytes read by the master.
- 'Transmit' mode is set when the master has started a master read transfer.

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
    );
}
```

4.2.22. Serial Peripheral Interface

1) R_SPI_Set

Synopsis

Configure the SPI pin selection.

Prototype

```
bool R_SPI_Set(
    uint32_t data      // Pin selection
);
```

Description

Set up the global SPI options.

[data]
Configure the SPI input and output pins. Use "|" to separate each selection.
Settings for RSPCKA, MOSIA and MISOA are compulsory.

- Pin selection

PDL_SPI_RSPCKA_PA5 or PDL_SPI_RSPCKA_PB0 or PDL_SPI_RSPCKA_PC5	Select the RSPCKA pin.
PDL_SPI_MOSIA_P16 or PDL_SPI_MOSIA_PA6 or PDL_SPI_MOSIA_PC6	Select the MOSIA pin.
PDL_SPI_MISOA_P17 or PDL_SPI_MISOA_PA7 or PDL_SPI_MISOA_PC7	Select the MISOA pin.
PDL_SPI_SSLA0_PA4 or PDL_SPI_SSLA0_PC4	Select the SSLA0 pin(optional).
PDL_SPI_SSLA1_PA0 or PDL_SPI_SSLA1_PC0	Select the SSLA1 pin (optional).
PDL_SPI_SSLA2_PA1 or PDL_SPI_SSLA2_PC1	Select the SSLA2 pin (optional).
PDL_SPI_SSLA3_PA2 or PDL_SPI_SSLA3_PC2	Select the SSLA3 pin (optional).

Return value

True if all parameters are valid; otherwise false.

Category

SPI

Reference

R_SPI_Create

Remarks

- Before calling R_SPI_Create, call this function to configure the relevant pins.
- Pins which are not used for the SPI functions may be omitted.
- Some pin options are not available on smaller device packages.

Program example

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

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

void func(void)
{
    /* Configure the applicable SPI pins */
    R_SPI_Set(
        PDL_SPI_RSPCKA_PA5 | PDL_SPI_MOSIA_PA6 |
          PDL_SPI_MISOA_PA7 |
         PDL_SPI_SSLA0_PA4 | PDL_SPI_SSLA1_PA0 |
         PDL_SPI_SSLA2_PA1 | PDL_SPI_SSLA3_PA2
    );
}
```

2) 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 control.
  If output signal SSLx (where x = 0, 1, 2 or 3) is used, call function R_SPI_Set to select the respective output pin.

PDL_SPI_PIN_SSL0_LOW or PDL_SPI_PIN_SSL0_HIGH or	Select active-low or active-high for output signal SSL0.
PDL_SPI_PIN_SSL1_LOW or PDL_SPI_PIN_SSL1_HIGH or	Select active-low or active-high for output signal SSL1.
PDL_SPI_PIN_SSL2_LOW or PDL_SPI_PIN_SSL2_HIGH or	Select active-low or active-high for output signal SSL2.
PDL_SPI_PIN_SSL3_LOW or PDL_SPI_PIN_SSL3_HIGH or	Select active-low or active-high 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

PDL_SPI_BUFFER_64 or **PDL_SPI_BUFFER_128**	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 Table 29.4 in the hardware manual).

Selection	Number of command transfers	Number of frames in each command transfer	Number of transfer frames
PDL_SPI_FRAME_1_1 or	1	1	1
PDL_SPI_FRAME_1_2 or	1	2	2
PDL_SPI_FRAME_1_3 or	1	3	3
PDL_SPI_FRAME_1_4 or	1	4	4
PDL_SPI_FRAME_2_1 or	2	1	2
PDL_SPI_FRAME_2_2 or	2	2	4
PDL_SPI_FRAME_3 or	3	1	3
PDL_SPI_FRAME_4 or	4	1	4
PDL_SPI_FRAME_5 or	5	1	5
PDL_SPI_FRAME_6 or	6	1	6
PDL_SPI_FRAME_7 or	7	1	7
PDL_SPI_FRAME_8	8	1	8

- Parity bit control

PDL_SPI_PARITY_NONE 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

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.

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_Set, R_SPI_Command

Remarks

- Function R_CGC_Set must be called (with the current clock source selected) before using this function.
- Function R_SPI_Set must be called before any use of this function.
- 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_FRAME_1_1,
        PDL_NO_DATA,
        2E6
    );
}
```

3) 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

None.

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
    );
}
```

4) 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

	Idle clock	Data sampling edge
PDL_SPI_CLOCK_MODE_0 or	Low	Rising
PDL_SPI_CLOCK_MODE_1 or		Falling
PDL_SPI_CLOCK_MODE_2 or	High	Rising
PDL_SPI_CLOCK_MODE_3		Falling

- Clock division

PDL_SPI_DIV_1 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

PDL_SPI_ASSERT_SSL0 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

PDL_SPI_SSL_NEGATE 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.

Description (2/2)

[data4]
Extended timing control. If multiple selections are required, use "|" to separate each selection. The default settings are shown in **bold**. For Slave mode, select PDL_NO_DATA.

- Extended timing selection

PDL_SPI_CLOCK_DELAY_MINIMUM or PDL_SPI_CLOCK_DELAY_EXTENDED	Select the minimum or extended delay between the assertion of the SSL pin and the start of RSPCK oscillation.

- SSL negation delay

PDL_SPI_SSL_DELAY_MINIMUM or PDL_SPI_SSL_DELAY_EXTENDED	Select the minimum or extended delay between the end of RSPCK oscillation and the negation of the active SSL pin.

- Next-access delay

PDL_SPI_NEXT_DELAY_MINIMUM or PDL_SPI_NEXT_DELAY_EXTENDED	Select the minimum or extended delay between the end of one frame and the start of the next frame.

Return value

True if all parameters are valid; otherwise false.

Category

SPI

Reference

R_SPI_Create

Remarks

- For Slave mode operation, configure command 0.
- When Clock-synchronous Slave mode is used, avoid selecting mode 0 or mode 2.
- If parity is enabled while in Master mode, both the frame data length and data transfer format should be the same for each 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 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
    );
}
```

5) R_SPI_Transfer

Synopsis

Transfer data over an SPI channel.

Prototype

```
bool R_SPI_Transfer(
    uint8_t data1,      // Channel selection
    uint8_t data2,      // DMAC / 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.

- DMAC / DTC trigger control

PDL_SPI_DMAC_DTC_TRIGGER_DISABLE or PDL_SPI_DMAC_TRIGGER_ENABLE or PDL_SPI_DTC_TRIGGER_ENABLE	Disable or enable activation of the DMAC or 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 DMAC or 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 DMAC or DTC shall be used to handle the data transfer.

[data5]
The number of times that the command sequence will be executed.
The value should not be zero if the DMAC and DTC trigger are disabled.

[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.
DMAC or DTC	The function to be called if an error has occurred, or when the DMAC or 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

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 DMAC / 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
    );
}
```

6) 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.

Category

SPI

Reference

R_SPI_Create

Remarks

- If a channel is disabled using PDL_SPI_DISABLE, call R_SPI_Create to resume channel operations.

Program example

```
/* 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
    );
}
```

7) 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

None.

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
    );
}
```

4.2.23. 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 PDL_CRC_POLY_CRC_16 or PDL_CRC_POLY_CRC_CCITT	$X^8 + X^2 + X + 1$ $X^{16} + X^{15} + X^2 + 1$ $X^{16} + X^{12} + X^5 + 1$

- 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

None.

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

PDL_CRC_CLEAR_RESULT 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

R_CRC_Create, R_CRC_Write

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
    );
}
```

4.2.24. 12-bit Analog to Digital Converter

1) R_ADC_12_Set

Synopsis

Select the I/O pins for the 12-bit ADC.

Prototype

```
bool R_ADC_12_Set(
    uint32_t data    // ADC unit selection
);
```

Description

Select the I/O pins for the12-bit ADC.

[data]
Select the pin set options. To set multiple options at the same time, use "|" to separate each value.

- Pin selection

PDL_ADC_12_PIN_AN000_P40	Select P40 for AN000.
PDL_ADC_12_PIN_AN001_P41	Select P41 for AN001.
PDL_ADC_12_PIN_AN002_P42	Select P42 for AN002.
PDL_ADC_12_PIN_AN003_P43	Select P43 for AN003.
PDL_ADC_12_PIN_AN004_P44	Select P44 for AN004.
PDL_ADC_12_PIN_AN005_P45	Select P45 for AN005.
PDL_ADC_12_PIN_AN006_P46	Select P46 for AN006.
PDL_ADC_12_PIN_AN007_P47	Select P47 for AN007.
PDL_ADC_12_PIN_AN008_PE0	Select PE0 for AN008.
PDL_ADC_12_PIN_AN009_PE1	Select PE1 for AN009.
PDL_ADC_12_PIN_AN010_PE2	Select PE2 for AN010.
PDL_ADC_12_PIN_AN011_PE3	Select PE3 for AN011.
PDL_ADC_12_PIN_AN012_PE4	Select PE4 for AN012.
PDL_ADC_12_PIN_AN013_PE5	Select PE5 for AN013.
PDL_ADC_12_PIN_AN014_PE6	Select PE6 for AN014.
PDL_ADC_12_PIN_AN015_PE7	Select PE7 for AN015.
PDL_ADC_12_PIN_ADTRG0_P07 or PDL_ADC_12_PIN_ADTRG0_P16 or PDL_ADC_12_PIN_ADTRG0_P25	Select P07, P16 or P25 for ADTRG0.

Return value

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

Category

12-bit ADC

Reference

R_ADC_12_CreateUnit

Remarks

- If there are I/O pins to be used, call this function before calling R_ADC_12_CreateUnit.
- Device packages with 64 or fewer pins do not have all of the pin options.

Program example

```
/* RPDL definitions */
#include "r_pdl_adc_12.h"

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

void func( void )
{
    /* Set analog channel AN000 */
    R_ADC_12_Set(
        PDL_ADC_12_PIN_AN000_P40
    );
}
```

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,     // Unit specific options
    uint32_t data3,     // Options for Group A
    uint32_t data4,     // Options for Group B
    double data5,       // Sampling time for internal reference voltage
    double data6,       // Pre-charging or discharging time for disconnection detection
    void * func1,       // Callback function for Group A
    uint8_t data7,      // Interrupt priority for Group A
    void * func2,       // Callback function for Group B
    uint8_t data8       // Interrupt priority for Group B
);
```

Description (1/4)

Set the ADC mode and operating condition.

[data1]
Select the ADC unit to be configured. This must always be 0.

[data2]
Conversion options. To set multiple options at the same time, use “|” to separate each value. The default settings are shown in **bold**.

- Input source

PDL_ADC_12_INPUT_AN or PDL_ADC_12_INPUT_REF	Select input from analog channels or the internal reference voltage.

- Scan mode

PDL_ADC_12_SCAN_SINGLE or PDL_ADC_12_SCAN_CONTINUOUS or PDL_ADC_12_SCAN_GROUP	Select single scan, continuous scan, or group scan mode.

- Trigger sources enable
  Not valid if PDL_ADC_12_SCAN_GROUP is selected for data2

PDL_ADC_12_ASYNC_TRIGGER_ENABLE or PDL_ADC_12_SYNC_TRIGGER_ENABLE	Enable the ADC to be started by asynchronous or synchronous trigger sources. If neither option is selected, only software will be used as the trigger source.

- Value addition control for internal reference voltage
  Not valid if PDL_ADC_12_INPUT_AN is selected for data2

PDL_ADC_12_REF_ADDITION_DISABLE or PDL_ADC_12_REF_ADDITION_ENABLE	Disable or enable value addition mode.

- Value addition count select

PDL_ADC_12_VALUE_ADDITION_0 or PDL_ADC_12_VALUE_ADDITION_1 or PDL_ADC_12_VALUE_ADDITION_2 or PDL_ADC_12_VALUE_ADDITION_3	Set the conversion to be 1-time, 2-time (addition once), 3-time(addition twice), or 4-time(addition three times)

- Data alignment

PDL_ADC_12_DATA_ALIGNMENT_RIGHT or PDL_ADC_12_DATA_ALIGNMENT_LEFT	The alignment of the 12-bit ADC conversion result within the 16-bit register. Ignored for channels using value addition mode (the 14-bit result is always left-aligned).

Description (2/4)

- Self-diagnostic control

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	Disable the self-diagnostic function, or enable and use the voltage on pin VREFH0: x 0, x ½, x 1 or automatically rotated voltage.

- Result register clearing

PDL_ADC_12_RETAIN_RESULT or PDL_ADC_12_CLEAR_RESULT	Retain or clear the value in each result register after it has been read.

- Disconnection detection assist function control

PDL_ADC_12_DDA_DISABLE or PDL_ADC_12_DDA_PRECHARGE or PDL_ADC_12_DDA_DISCHARGE	Disable the disconnection detection assist function, or enable and set it to pre-charge or discharge mode

- Sampling time calculation for internal reference voltage

PDL_ADC_12_ADSSTR_CALCULATE or PDL_ADC_12_ADSSTR_SPECIFY	Select whether parameter data5 is used to calculate the ADSSTR value, or contains the value to be stored in register ADSSTR.

- Pre-charging or discharging time calculation for disconnection detection

PDL_ADC_12_ADDISCR_CALCULATE or PDL_ADC_12_ADDISCR_SPECIFY	Select whether parameter data6 is used to calculate the ADDISCR value, or contains the value to be stored in register ADDISCR. Valid unless PDL_ADC_12_DDA_DISABLE is selected

Description (3/4)

[data3] [data4]
Trigger source selection. To set multiple options at the same time, use "|" to separate each value.

- Trigger selection
  Valid only if PDL_ADC_12_SYNC_TRIGGER_ENABLE is selected.

PDL_ADC_12_GP_TRIGGER_MTU_TRG0AN or	TRGA input capture/compare match A from MTU0.
PDL_ADC_12_GP_TRIGGER_MTU_TRG0BN or	TRGB input capture/compare match B from MTU0.
PDL_ADC_12_GP_TRIGGER_MTU_TRGAN or	TRGA input capture/compare match or MTU4.TCNT underflow (trough) in complementary PWM mode from MTU0 to MTU4.
PDL_ADC_12_GP_TRIGGER_MTU_TRG0EN or	TRGE compare match from MTU0.
PDL_ADC_12_GP_TRIGGER_MTU_TRG0FN or	TRGF compare match from MTU0.
PDL_ADC_12_GP_TRIGGER_MTU_TRG4AN or	MTU4.TADCORA and MTU4.TCNT compare match (interrupt skipping function 1).
PDL_ADC_12_GP_TRIGGER_MTU_TRG4BN or	MTU4.TADCORB and MTU4.TCNT compare match (interrupt skipping function 1).
PDL_ADC_12_GP_TRIGGER_MTU_TRG4ABN or	MTU4.TADCORA and MTU4.TCNT compare match and MTU4.TADCORB and MTU4.TCNT compare match (interrupt skipping function 1).
PDL_ADC_12_GP_TRIGGER_ELC	Trigger from the ELC.

- DTC/DMAC trigger control

PDL_ADC_12_GP_DMAC_DTC_TRIGGER_DISABLE or PDL_ADC_12_GP_DMAC_TRIGGER_ENABLE or PDL_ADC_12_GP_DTC_TRIGGER_ENABLE	Disable or enable activation of the DMAC or DTC.

[data5]
The data to be used for the sampling state register value calculations for self-diagnosis or internal reference voltage, depending on the input source parameter of data2. If PDL_ADC_12_ADSSTR_SPECIFY is selected for parameter data2, the value should not be less than 12 or more then 255.

Data use	Parameter type
The timer period in seconds or	double
The value to be put in register ADSSTR	uint8_t

[data6]
The data to be used for the disconnecting detection control register value calculations. If PDL_ADC_12_ADDISCR_SPECIFY is selected for data2, the value should not be 0 or more than 15.

Data use	Parameter type
The timer period in seconds or	double
The value to be put in register ADDISCR	uint8_t

[func1]
The function to be called when the ADC conversion scan cycle is complete in single scan mode and continuous scan mode; or when the ADC conversion scan cycle is complete for Group A in group scan mode.
Specify PDL_NO_FUNC if no callback function is required.

Description (4/4)

[data7]
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 func1.

[func2]
The function to be called when the ADC conversion scan cycle is complete for Group B in group scan mode.
Specify PDL_NO_FUNC if no callback function is required.

[data8]
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 func2.

Return value

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

Category

12-bit ADC

References

R_CGC_Set

Remarks

- Interrupts are enabled automatically if a callback function is specified.
  Please see the notes on callback function usage in §6.
- If an external trigger is used, the low-level pulse width must be at least 1.5 PCLKD cycles.
- This function brings the converter unit out of the power-down state.
- 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.
- Function R_CGC_Set must be called (with the current clock source selected) before using this function.
- In group scan mode or continuous scan mode, PDL_ADC_12_INPUT_REF cannot be selected.
- When the internal reference voltage is to be converted, the disconnection detection assist function is not available.
- Allow 1µs to elapse from the completion of this function to the start of the first conversion.
- Make sure sampling time calculated or specified for channel 0 and self-diagnosis are the same.
- For more details of trigger sources, please refer to the RX220 hardware manual.

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 in single mode */
    R_ADC_12_CreateUnit(
        0,
        PDL_ADC_12_SCAN_SINGLE,
        PDL_ADC_12_GP_TRIGGER_MTU_TRG0AN,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        ADCIntFunc,
        2,
        PDL_NO_FUNC,
        PDL_NO_DATA
    );
}
```

3) R_ADC_12_CreateChannel

Synopsis

Configure 12-bit ADC analog channels.

Prototype

```
bool R_ADC_12_CreateChannel(
    uint8_t data1,     // ADC unit selection
    uint8_t data2,     // Analog channel selection
    uint16_t data3,    // Channel configuration
    double data4       // Sampling time
);
```

Description

Channel specific control. Used to complement R_ADC_12_CreateUnit to configure 12-bit ADC analog channels, if analog channels are selected as the input source.

[data1]
Select the ADC unit. This must always be 0.

[data2]
Select the analog input channel. This must be from 0 to 15.

[data3]
Channel options. To set multiple options at the same time, use "|" to separate each value.
The default settings are shown in **bold**.

- Group selection

PDL_ADC_12_CH_GROUP_A or PDL_ADC_12_CH_GROUP_B	Assign the channel to Group A or Group B

- Value addition control

PDL_ADC_12_CH_VALUE_ADDITION_DISABLE or PDL_ADC_12_CH_VALUE_ADDITION_ENABLE	Enable or disable value addition

- Double trigger control

PDL_ADC_12_CH_DOUBLE_TRIGGER_DISABLE or PDL_ADC_12_CH_DOUBLE_TRIGGER_ENABLE	Enable or disable double trigger

- Sampling time calculation

PDL_ADC_12_CH_ADSSTR_CALCULATE or PDL_ADC_12_CH_ADSSTR_SPECIFY	Select whether parameter data4 is used to calculate the ADSSTR value, or contains the value to be stored in register ADSSTR.

[data4]
The data to be used for the sampling state register value calculations.
If PDL_ADC_12_CH_ADSSTR_SPECIFY is selected for data3, the value should not be less than 12 or more then 255.

Data use	Parameter type
The timer period in seconds or	double
The value to be put in register ADSSTR	uint8_t

Return value

True if a valid unit is selected; otherwise false.

Category

12-bit ADC

Reference

R_ADC_12_CreateUnit

Remarks

- If analog channels are used as the input sources, call this function after calling R_ADC_12_CreateUnit.
- Function R_CGC_Set must be called (with the current clock source selected) before using this function.
- Make sure no more than 1 channel is configured with the parameter of PDL_ADC_12_CH_DOUBLE_TRIGGER_ENABLE.
- Channels 8 -15 share the same ADSSTR register. Once set sampling time for each of them, the same setting applies to all the other channels. A later setting overwrites the previous one.
- Make sure sampling time calculated or specified for channel 0 and self-diagnosis are the same.

Program example

```
/* RPDL definitions */
#include "r_pdl_adc_12.h"

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

void func( void )
{
    /* Configure AN000 */
    R_ADC_12_CreateChannel(
        0,
        0,
        PDL_ADC_12_CH_GROUP_A |
         PDL_ADC_12_CH_ADSSTR_CALCULATE,
        5E-6
    );
}
```

4) R_ADC_12_Destroy

Synopsis

Shut down the ADC unit.

Prototype

```
bool R_ADC_12_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 to be shut down. This must always be 0.

Return value

True if a valid unit is selected; otherwise false.

Category

12-bit ADC

Reference

None

Remarks

- This function includes 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 the ADC unit */
    R_ADC_12_Destroy(
        0
    );
}
```

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

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_12_0_ON or	Start a software-triggered conversion or re-enable the trigger.
PDL_ADC_12_0_OFF	Stop the conversion (and disable all triggers).

- Control the CPU during the ADC conversion.

PDL_ADC_12_CPU_OFF	Stop the CPU when the scan 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

None

Remarks

- For single scan mode, the ADC will stop automatically when the conversion is complete.
- Do not select CPU Off unless there is any interrupt to wake up the CPU.

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
    );
}
```

6) R_ADC_12_Read

Synopsis

Read the ADC conversion results.

Prototype

```
bool R_ADC_12_Read(
    uint8_t data1,      // ADC unit selection
    uint16_t * data2,   // Pointer to the address where the results are to be stored
    uint16_t * data3    // Pointer to the address where the result is to be stored
);
```

Description

Reads the conversion values for an ADC unit.

[data1]
Select the ADC unit to be configured. This must always be 0.

[data2]
Specify a pointer to an array with 16 members, where the converted values for analog input channels are to be stored.

[data3]
Specify a pointer to the address where the converted value for internal reference voltage, diagnostic result or double trigger result is to be stored.
Refer to hardware manual Section 32.2.4 for the format of self-diagnosis result.

Return value

True if a valid unit is selected; otherwise false.

Category

12-bit ADC

Reference

R_ADC_12_CreateUnit, R_ADC_12_CreateChannel

Remarks

- From 1 to 16 valid conversion results will be read and stored to data2. The number depends on the parameters supplied to R_ADC_12_CreateUnit and R_ADC_12_CreateChannel for configuration.
- The data alignment is controlled using the R_ADC_12_CreateUnit function.
- If the internal reference voltage is selected as the input source, valid pointer should be supplied to data3, while data2 will be ignored.
- If analog channels are selected as the input source, valid array pointer should be supplied to data2. If double trigger or self-diagnostic is enabled, the respective result will be stored to data3, if a valid pointer is supplied.
- If no callback function is used, this function waits for the IR flags 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.

Program example

```
/* RPDL definitions */
#include "r_pdl_adc_12.h"

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

void func(void)
{
    uint16_t ADCresult[16];
    uint16_t DIAGresult;

    /* Read the ADC */
    R_ADC_12_Read(
        0,
        ADCresult,
        &DIAGresult
    );
}
```

4.2.25. Comparator A

1) R_CPA_Create

Synopsis

Configure the Comparator A module.

Prototype

```
bool R_CPA_Create(
        uint8_t data1,      // Comparator A channel selection.
        uint16_t data2,     // Configuration for Comparator A
        void * func,        // Callback function pointer for Comparator A
        uint8_t data3       // Interrupt priority
);
```

Description

Set up Comparator A; enable the interrupt, register and callback functions.

[data1]
The comparator A channel number n (where n = 0 to 1).

[data2]
Configure the Comparator A channel.
To set multiple options at the same time, use "|" to separate each value.
The default settings are shown in **bold**. Specify PDL_NO_DATA to use the defaults.

- Operation

PDL_CPA_MONITOR_ONLY or PDL_CPA_INTERRUPT_RESET_ENABLE	Select no action, for monitor only enable interrupt/reset for comparator A

- Digital filter

PDL_CPA_FILTER_DISABLE or PDL_CPA_FILTER_LOCO_DIV_1 or PDL_CPA_FILTER_LOCO_DIV_2 or PDL_CPA_FILTER_LOCO_DIV_4 or PDL_CPA_FILTER_LOCO_DIV_8	Configure the digital filter for comparator A

- Mode Select. Not valid for monitor-only operation selected

PDL_CPA_MODE_SELECT_INTERRUPT or PDL_CPA_MODE_SELECT_RESET	Select mode for comparator A

- Reset negation Select. Valid for reset mode selected

PDL_CPA_NEGATION_AFTER_RESET or PDL_CPA_NEGATION_AFTER_DETECT	Select reset negation for comparator A

- Interrupt/ Event Generation Condition Select. Valid for interrupt mode selected

PDL_CPA_IRQ_BELOW_CVREFA or PDL_CPA_IRQ_ABOVE_N_EQUAL_CVREFA or PDL_CPA_IRQ_CROSS_CVREFA	Select Interrupt/ Event Generation Condition of above and equal, below or cross CVREFA for comparator A

- Interrupt Type. Valid for interrupt mode selected

PDL_CPA_NONMASKABLE_INTERRUPT or PDL_CPA_MASKABLE_INTERRUPT	Select non–maskable interrupt Select maskable interrupt or ELC event

[func]
The function to be called when a maskable interrupt request from comparator A, except ELC is used.
Specify PDL_NO_FUNC if PDL_CPA_NONMASKABLE_INTERRUPT, ELC Event Generation Condition Selection or monitor-only operation are selected

[data3]
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

Comparator A

Reference

R_CGC_Set, R_LVD_Create, R_LVD_Control and R_CPA_GetStatus

Remarks

- Do not use VDET1 and Comparator A channel 0, VDET2 and comparator A channel 1 at same time because they share same registers. See R_LVD_Create and R_LVD_Control
- Function R_CGC_Set must be called (with the current clock source selected) before using this function.
- If using the digital filter, the LOCO clock must be enabled, see R_CGC_Set.
- Disable the digital filter circuit when using Comparator A interrupt to return from software standby mode.
- Use R_CPA_GetStatus to determine the interrupt cause.
- Do not select PDL_CPA_NEGATION_AFTER_RESET if a transition to software standby is to be made.
- Must call R_INTC_CreateExtInterrupt(PDL_INTC_LVDi_ENABLE) to enable the NMI before set PDL_CPA_INTERRUPT_NONMASKABLE.
  Note: Comparator Non-Interrupt Callback function is created by this call.
- Set PDL_CPA_NEGATION_AFTER_RESET under the LOCO in operating.
- The LVDi reset or LVDi non-maskable interrupt should not be generated during flash memory programming/erasure.
- If a callback function is specified, this function will enable the relevant interrupt.
  Please see the notes on callback function use in §6.
- To enable the CPA event link output function, enable the CPA first then enable the CPA event link function at the ELC. To disable this function, disable the CPA event link function at the ELC first then disable the CPA.

Program example

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

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

void CPAi_handler(void){}
void NMI_handler_cpa(void){}

void func(void)
{
    /* Configure the NMI pin */
    R_INTC_CreateExtInterrupt(
        PDL_INTC_NMI,
        PDL_INTC_FALLING|PDL_INTC_LVD1_ENABLE,
        NMI_handler_cpa,
        7
    );

    /* Non-Maskable Interrupt Comparator A channel 0 */
    /* Digital Filter, interrupt enable, LOCO_div_2 */
    R_CPA_Create(
        0,
        PDL_CPA_FILTER_LOCO_DIV_2 |
         PDL_CPA_INTERRUPT_RESET_ENABLE,
        PDL_NO_FUNC,
        0
    );

    /* Maskable Interrupt Comparator A channel 1 */
    /* Digital Filter, interrupt enable, LOCO_div_1 */
    R_CPA_Create(
        1,
        PDL_CPA_FILTER_LOCO_DIV_1 |
         PDL_CPA_INTERRUPT_RESET_ENABLE |
         PDL_CPA_IRQ_CROSS_CVREFA |
         PDL_CPA_MASKABLE_INTERRUPT,
        CPAi_handler,
        7
    );
}
```

2) R_CPA_Control

Synopsis

Control the Comparator A module.

Prototype

```
bool R_CPA_Control (
        uint8_t data1,      // Comparator selection
        uint8_t data2       // Settings for Comparator
);
```

Description

Control Comparator A1 or A2.

[data1]
- The comparator A channel number n (where n = 0 to 1).

[data2]
Disable the Comparator A.

- Comparator circuit option. Valid for interrupt and reset mode.

PDL_CPA_LVD_CIRCUIT_DISABLE	Disable comparator A circuit.

Return value

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

Category

Comparator A

Reference

R_CPA_Create, R_LVD_Create and R_LVD_Control

Remarks

- This function should not be called before R_CPA_Create.
- Do not use VDET1 module and Comparator A channel 0, VDET2 and comparator A channel 1 at same time because they share the same registers. See R_LVD_Create and R_LVD_Control.
- The LVDi reset or LVDi non-maskable interrupt should not be generated during flash memory programming/erasure.

Program example

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

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

void func(void)
{
    /* Disable Comparator A channel 0 */
    R_CPA_Control(
        0,
        PDL_CPA_LVD_CIRCUIT_DISABLE
    );
}
```

3) R_CPA_GetStatus

Synopsis

Check the status of the Comparator A module.

Prototype

```
bool R_CPA_GetStatus(
        uint8_t * data      // Status flags pointer
);
```

Description

Return the status flags.

[data]
The comparator A status flag shall be stored in the following format.

b7 - b6	b5	b4	b3 - b2	b1	b0
	Status	Change		Status	Change
0	0: CMPA2 < CVREFA 1: CMPA2 ≥ CVREFA	0: None 1: Detected	0	0: CMPA1 < CVREFA 1: CMPA1 ≥ CVREFA	0: None 1: Detected

Return value

True.

Category

Comparator A

Reference

None

Remarks

- If the voltage change detection flag is set it is automatically cleared by this function.

Program example

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

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

void func(void)
{
    uint8_t StatusFlags;

    /* Read the CPA status */
    R_CPA_GetStatus(
        &StatusFlags
    );
}
```

4.2.26. Data Operation Circuit

1) R_DOC_Create

Synopsis

Configure the Data Operation Circuit.

Prototype

```
bool R_DOC_Create(
    uint8_t data1,      // Configuration
    uint16_t data2,     // Output value
    void* func,         // Callback function
    uint8_t data3       // Interrupt priority level
);
```

Description

Enable the DOC module and set the operating conditions.

[data1]
Operation Mode

PDL_DOC_COMPARISON_MATCH or PDL_DOC_COMPARISON_MISMATCH or PDL_DOC_MODE_ADD or PDL_DOC_MODE_SUBTRACT	Specify the mode of operation.

[data2]
This meaning of this parameter depends upon the Operation Mode:

Operation Mode	Description
Comparison	The comparison value.
Addition	The initial output value before any additions are made.
Subtraction	The initial output value before any subtractions are made.

[func]
The function to be called when a DOC interrupt is generated.
Specify PDL_NO_FUNC if no callback function is required.

[data3]
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 func.

Return value

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

Category

DOC

References

None.

Remarks

- In Addition Mode an interrupt is generated if the result of the addition exceeds FFFFh.
- In Subtraction Mode an interrupt is generated if the result of the subtraction is less than zero.
- In Comparison Mode an interrupt is generated when the comparison criteria (Match or Mismatch) is met.
- This function brings the DOC module out of the power-down state.
- If a callback function is specified then interrupts will be automatically enabled.
- After calling a callback function the DOC flag is automatically cleared.

Program example

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

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

void Callback(void);

void func(void)
{
    /* Setup DOC in addition mode */
    R_DOC_Create(
        PDL_DOC_MODE_ADD,
        0,
        Callback,
        15
    );
}
```

2) R_DOC_Destroy

Synopsis	Disable the Data Operation Circuit.
Prototype	**bool R_DOC_Destroy(** **void** **);**
Description	Disable and enable the DOC module stop state.
Return value	True
Category	DOC
Reference	None
Remarks	None
Program example	

```
#include "r_pdl_doc.h"

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

void func(void)
{
    R_DOC_Destroy();
}
```

3) R_DOC_Control

Synopsis

Control the Data Operation Circuit.

Prototype

```
bool R_DOC_Control(
    uint8_t data1,      // Configuration
    uint16_t data2      // Data
);
```

Description

Control the DOC Module.

[data1]
Control operation. To set multiple options at the same time, use "|" to separate each value. If no selection is made specify PDL_NO_DATA, the control setting will be left unchanged.

- Operation Mode

PDL_DOC_COMPARISON_MATCH or PDL_DOC_COMPARISON_MISMATCH or PDL_DOC_MODE_ADD or PDL_DOC_MODE_SUBTRACT	If required, specify a new mode of operation to change to.

- DOC Flag

PDL_DOC_FLAG_CLEAR	Clear the DOC flag. If this flag is set when interrupts are enabled an interrupt will be generated. Note: The DOC flag is automatically cleared when the callback function is called.

- Interrupt control

PDL_DOC_INTERRUPT_ENABLE or PDL_DOC_INTERRUPT_DISABLE	Enable or disable the DOC interrupt.

- Update the DOC data value.

PDL_DOC_DATA_UPDATE	Update the DOC with the value specified in data2. See data2 description for meaning.

[data2]
This meaning of this parameter depends upon the Operation Mode:

Operation Mode	Description
Comparison	The comparison value.
Addition	The initial output value, before additions are made.
Subtraction	The initial output value, before subtractions are made.

Return value

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

Category

DOC

References

R_DOC_Create

Remarks

- Interrupts can only be enabled if a callback was registered using R_DOC_Create.

Program example

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

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

void func(void)
{
    /* Change to subtraction mode with initial value 500 */
    R_DOC_Control(
        PDL_DOC_MODE_SUBTRACT | PDL_DOC_DATA_UPDATE,
        500
    );
}
```

4) R_DOC_Read

Synopsis

Read the Data Operation Circuit result.

Prototype

```
bool R_DOC_Read(
    uint8_t * data1,     // Pointer to status storage location
    uint16_t * data2     // Pointer to value storage location.
);
```

Description

Read the DOC status and output.

[data1]
The status flags shall be stored in the format below.
Specify PDL_NO_PTR if this information is not required.

b7 - b1	b0
-	Flag (see remarks)

[data2]
This meaning of this parameter depends upon the Operation Mode as specified in the table below. Specify PDL_NO_PTR if this information is not required.

Operation Mode	Description
Comparison	The set comparison value.
Addition	The addition result.
Subtraction	The subtraction result.

Return value

True

Category

DOC

References

None.

Remarks

- In Addition Mode the flag is set if the result of the addition exceeds FFFFh.
- In Subtraction the flag is set if the result of the subtraction is less than zero.
- In Comparison Mode the flag is set when the comparison criteria (Match/Mismatch) is met.
- If the flag is set it is automatically cleared by this function.
- If using interrupts the flag is automatically cleared when the interrupt is handled.

Program example

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

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

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

    /* Read result */
    R_DOC_Read(
        &status,
        &result
    );
}
```

5) R_DOC_Write

Synopsis

Write data to the Data Operation Circuit.

Prototype

```
bool R_DOC_Write(
    uint16_t * data1,    // Pointer to buffer holding data to write.
    uint16_t * data2     // Number of 16-bit words to write.
);
```

[data1]
The start address of the data to be written.

[data2]
The number of 16-bit words to write.

Return value

True

Category

DOC

References

None.

Remarks

- This function will not return until all the supplied data has been written to the DOC.
- The DMAC/DTC can be used to write data to the DOC independently of this function.

Program example

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

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

void func(void)
{
    uint16_t data[10] = {1,2,3,4,5,6,7,8,9,10};

    /* Write 10 numbers to the DOC */
    R_DOC_Write(
        data,
        10
    );
}
```

5. Usage Examples

This chapter shows programming examples for each driver in this library.

5.1. Clock Generation Circuit

Figure 3 shows an example of configuring the clock generation circuit.

After a power-on reset, the main clock oscillator is switched off.
The MCU is using the LOCO as the clock source.
The calls to R_CGC_Set configure the LOCO dividers and enable the main clock oscillator.
R_CGC_Control is used to select the Main clock as the clock source.

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

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

void main(void)
{
    /* Set the LOCO clock settings (the clock source used after a power-on reset) */
    /* ICLK = 125 kHz, PCLKB = 125 kHz, FCLK = 125 kHz */
    R_CGC_Set(
        PDL_CGC_CLK_LOCO,
        PDL_NO_DATA,
        125E3,
        125E3,
        PDL_NO_DATA,
        125E3,
        125E3
    );

    /* Configure main clock operation using an external 20.0 MHz clock */
    /* ICLK = 20 MHz, PCLKD = 20/4 MHz, PCLKB = 20/8 MHz, FCLK = 20/16 MHz */
    R_CGC_Set(
        PDL_CGC_CLK_MAIN,
        PDL_CGC_MAIN_RESONATOR,
        20E6,
        20E6,
        20E6 / 4,
        20E6 / 8,
        20E6 / 16
    );

    /* Select the Main clock as the clock source */
    R_CGC_Control(
        PDL_CGC_CLK_MAIN,
        PDL_NO_DATA,
        PDL_NO_DATA
    );
}
```

Figure 3: Example of Clock configuration and control

5.2. Interrupt control

Figure 4 shows an example of external interrupt use.

Pin IRQ1 on port pin P31 is used to detect a falling edge and generates an interrupt. The interrupt handler inverts the edge detection and disables further interrupts. Pin IRQ3 on port pin P33 is used to detect a falling edge and generates an interrupt. Pin IRQ4 on port pin P34 is used to detect low level state and generates an interrupt.

```
/* Peripheral driver function prototypes */
#include "r_pdl_intc.h"

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

/* Callback function prototypes */
static void SW1_handler(void);
static void SW2_handler(void){}
static void SW3_handler(void){}

void main(void)
{
    /* Set the CPU's Interrupt Priority Level to 0 */
    R_INTC_Write(
        PDL_INTC_REG_IPL,
        0
    );

    /* Select the pins for SW1, SW2 and SW3 */
    R_INTC_SetExtInterrupt(
        PDL_INTC_IRQ1_P31 | PDL_INTC_IRQ3_P33 | PDL_INTC_IRQ4_P34
    );

    /* Configure the SW1 interrupt */
    R_INTC_CreateExtInterrupt(
        PDL_INTC_IRQ1,
        PDL_INTC_FALLING,
        SW1_handler,
        7
    );

    /* Configure the SW2 interrupt */
    R_INTC_CreateExtInterrupt(
        PDL_INTC_IRQ3,
        PDL_INTC_FILTER_DIV_32 | PDL_INTC_FALLING,
        SW2_handler,
        7
    );

    /* Configure the SW3 interrupt */
    R_INTC_CreateExtInterrupt(
        PDL_INTC_IRQ4,
        PDL_INTC_LOW,
        SW3_handler,
        7
    );

    while(1);
}

static void SW1_handler(void)
{
    uint8_t irq_status = 0u;

    R_INTC_GetExtInterruptStatus(
        PDL_INTC_IRQ1,
        &irq_status
    );
```

```
        /* Falling edge detected? */
        if ((irq_status & 0x0C) == 0x04)
        {
            /* Disable and invert the edge interrupt */
            R_INTC_ControlExtInterrupt(
                PDL_INTC_IRQ1,
                PDL_INTC_RISING | PDL_INTC_DISABLE
            );
        }
        else if ((irq_status & 0x0C) == 0x08)
        {
            /* Disable and invert the edge interrupt */
            R_INTC_ControlExtInterrupt(
                PDL_INTC_IRQ1,
                PDL_INTC_FALLING | PDL_INTC_DISABLE
            );
        }
    }
```

Figure 4: Example of External Interrupt

5.3. I/O Port

Figure 5 shows examples of I/O port configuration, reading and writing.

```
/* Peripheral driver function prototypes */
#include "r_pdl_io_port.h"

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

void main(void)
{
    uint8_t result;
    uint16_t direction;

    /* Set all reserved I/O port pins to the recommended state */
    R_IO_PORT_NotAvailable();

    /* Configure port 4 as an input */
    R_IO_PORT_Set(
        PDL_IO_PORT_4_0 | PDL_IO_PORT_4_1 | \
        PDL_IO_PORT_4_2 | PDL_IO_PORT_4_3 | \
        PDL_IO_PORT_4_4 | PDL_IO_PORT_4_5 | \
        PDL_IO_PORT_4_6 | PDL_IO_PORT_4_7,
        PDL_IO_PORT_INPUT
    );

    /* Configure port pin P30 as an open-drain output */
    R_IO_PORT_Set(
        PDL_IO_PORT_3_0,
        PDL_IO_PORT_OUTPUT | PDL_IO_PORT_TYPE_NMOS
    );

    /* Read the value of all the pins on port 4 */
    R_IO_PORT_Read(
        PDL_IO_PORT_4,
        &result
    );

    /* Set pin P30 to output high */
    R_IO_PORT_Write(
        PDL_IO_PORT_3_0,
        1
    );

    /* Invert pin P30 */
    R_IO_PORT_Modify(
        PDL_IO_PORT_3_0,
        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 P1 */
    R_IO_PORT_ReadControl(
        PDL_IO_PORT_1,
        PDL_IO_PORT_DIRECTION,
        &direction
    );

    /* 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: Examples of I/O Port Operations

5.4. MCU Operation

Figure 6 shows an example of MCU usage. It detects if a Cold start has occurred.

```
/* Peripheral driver function prototypes */
#include "r_pdl_mcu.h"

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

void main(void)
{
        uint16_t mode_status;
        uint16_t reset_status;

        /* Read the MCU status registers */
        R_MCU_GetStatus(
                &mode_status,
                &reset_status,
                PDL_NO_PTR,
                PDL_NO_PTR
        );

        /* Cold start? */
        if ((reset_status & BIT_8) == 0)
        {
                /* Set the warm start indicator */
                R_MCU_Control(
                        PDL_MCU_WARM_START
                );
        }

        /* Reset the MCU */
        R_MCU_Control(
                PDL_MCU_RESET_START
        );

        while(1);
}
```

Figure 6: Example of MCU operation

5.5. Voltage Detection Circuit

5.5.1. Maskable interrupts

This shows an example of Voltage detection circuit usage.
If the supply voltage drops below 2.8V, the callback function is called.

```
/* Peripheral driver function prototypes */
#include "r_pdl_lvd.h"

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

static void Callback_LVD(void);

void main(void)
{
    /* Initialise the system clocks */
    NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.

    /* Setup VDET2 to call a function if VCC drops below 3.1V */
    R_LVD_Create(
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_LVD_INTERRUPT_MI_DETECT_FALL |PDL_LVD_FILTER_LOCO_DIV_2 | \
            PDL_LVD_VDET2_PIN_VCC,
        PDL_LVD_VOLTAGE_LEVEL_310,
        PDL_NO_FUNC,
        PDL_NO_DATA,
        Callback_LVD,
        15
    );
}

/* Low Voltage Callback function */
static void Callback_LVD(void)
{
    uint8_t status;
    /* Read status */
    R_LVD_GetStatus(&status);

    /*User: Handle Low Voltage Detection */
}
```

Figure 7: Example of Voltage Detection Circuit use

5.5.2. Non-maskable interrupts

This shows an example of Voltage detection circuit usage
An NMI is generated if the supply voltage drops below 3.1V.

```
/* Peripheral driver function prototypes */
#include "r_pdl_lvd.h"
#include "r_pdl_intc.h"

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

static void Callback_NMI(void);

void main(void)
{
    /* Initialise the system clocks */
    NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.

    /* Configure the NMI to be triggered by the LVD1 signal only (no NMI pin) */
    R_INTC_CreateExtInterrupt(
        PDL_INTC_NMI,
        PDL_INTC_LVD1_ENABLE,
        Callback_NMI,
        PDL_NO_DATA
    );

    /* Setup VDET1 to callback if VCC drops below 3.1V */
    R_LVD_Create(
        PDL_LVD_INTERRUPT_NMI_DETECT_FALL | PDL_LVD_FILTER_DISABLE,
        PDL_LVD_VOLTAGE_LEVEL_310,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_FUNC,
        PDL_NO_DATA,
        PDL_NO_FUNC,
        PDL_NO_DATA
    );
}
/* NMI Callback function */
static void Callback_NMI(void)
{
    uint8_t status;

    /* Read the NMI status */
    R_INTC_GetExtInterruptStatus(
        PDL_INTC_NMI,
        &status
    );

    /* Did an LVD1 trigger occur */
    if ((status & BIT_6) != 0)
    {
        /* Clear the LVD monitor 1 flag */
        R_LVD_Control(
            PDL_LVD_CLEAR_DETECTION,
            PDL_NO_DATA
        );

        /* Clear the NMI LVD1 flag */
        R_INTC_ControlExtInterrupt(
            PDL_INTC_NMI,
            PDL_INTC_CLEAR_LVD1_FLAG
        );
    }
}
```

Figure 8: Example of Voltage Detection Circuit use

5.6. Clock Frequency Accuracy Measurement Circuit

Figure 9 shows an example of clock frequency measurement usage.

The main clock is used as the reference, to measure the IWDT LOCO frequency.

```
/* Peripheral driver function prototypes */
#include "r_pdl_cac.h"
#include "r_pdl_cgc.h"

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

/* Callback functions */
void CAC_frequency_error( void );
void CAC_measurement_complete( void );
void CAC_overflow( void );

void main(void)
{
    /* Configure the IWDTLOCO clock settings */
    R_CGC_Set(
        PDL_CGC_CLK_IWDTLOCO,
        PDL_NO_DATA,            /*Reserved*/
        125000,                 /*IWDTLOCO*/
        PDL_NO_DATA,            /*ICLK*/
        PDL_NO_DATA,            /*PCLKD*/
        PDL_NO_DATA,            /*PCLKB*/
        PDL_NO_DATA             /*FCLK*/
    );

    /* Configure the main clock settings */
    R_CGC_Set(
        PDL_CGC_CLK_MAIN,
        PDL_CGC_MAIN_RESONATOR,
        20E6,                           /* Main */
        20E6,                           /* ICLK */
        20E6,                           /* PCLKD */
        20E6,                           /* PCLKB */
        20E6                            /* FCLK */
    );

    /* Select the main Clock as the clock source and enable the IWDT Clock */
    R_CGC_Control(
        PDL_CGC_CLK_MAIN,
        PDL_CGC_MAIN_ENABLE,
        PDL_CGC_IWDTLOCO_ENABLE
    );

    /* Use the main clock to check the IWDTLOCO accuracy (±10%) */
    R_CAC_Create(
        PDL_CAC_REFERENCE_MAIN |
            PDL_CAC_REFERENCE_RISING |
            PDL_CAC_REFERENCE_DIV_8192 |
            PDL_CAC_MEASURE_IWDTLOCO |
            PDL_CAC_MEASURE_DIV_1 |
            PDL_CAC_LIMIT_TOLERANCE,
        PDL_NO_DATA, /*Not using CACREF pin*/
        PDL_NO_DATA, /*Not using CACREF pin*/
        10,             /*10% tolerence*/
        10,             /*10% tolerence*/
        CAC_frequency_error,
        5,              /*Priority */
        CAC_measurement_complete,
        6,              /*Priority */
        CAC_overflow,
```

```
            10              /*Priority*/
        );

        while(1);
    }

    /* Frequency error callback */
    void CAC_frequency_error( void )
    {
        uint8_t Status_flags;
        uint16_t upper_limit;
        uint16_t lower_limit;
        uint16_t counter;

        R_CAC_GetStatus(
            &Status_flags,
            &upper_limit,
            &lower_limit,
            &counter
        );
        /* Clear the error flag */
        R_CAC_Control(
            PDL_CAC_CLEAR_FREQUENCY_ERROR,
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_DATA
        );
        /* TODO: Handle the frequency error. */
    }

    void CAC_measurement_complete( void )
    {
        uint8_t Status_flags;
        uint16_t upper_limit;
        uint16_t lower_limit;
        uint16_t counter;

        R_CAC_GetStatus(
            &Status_flags,
            &upper_limit,
            &lower_limit,
            &counter
        );
        /* Clear the measurement-complete flag and stop the CAC */
        R_CAC_Control(
            PDL_CAC_DISABLE | PDL_CAC_CLEAR_MEASUREMENT,
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_DATA
        );
    }

    /* Overflow callback */
    void CAC_overflow( void )
    {
        /* Clear the overflow flag */
        R_CAC_Control(
            PDL_CAC_CLEAR_OVERFLOW,
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_DATA
        );
        /* TODO: Handle the overflow error. */
    }
```

Figure 9: Example of Clock Frequency measurement use

5.7. Low Power Consumption

5.7.1. Software Standby Mode

Figure 10 shows an example of entering Software Standby mode through Low Power Consumption control.

```
/* Peripheral driver function prototypes */
#include "r_pdl_cgc.h"
#include "r_pdl_lpc.h"
#include "r_pdl_intc.h"

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

static void SW1_handler(void);

void main(void)
{
    /* Set Switch1 (SW1) interrupt */
    R_INTC_SetExtInterrupt(
        PDL_INTC_IRQ1_P31
    );

    /* Enable the switch SW1 interrupt */
    R_INTC_CreateExtInterrupt(
        PDL_INTC_IRQ1,
        PDL_INTC_FALLING,
        SW1_handler,
        15
    );

    /* Select the default options */
    R_LPC_Create(
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA
    );

    /* Enter software standby mode */
    R_LPC_Control(
        PDL_LPC_MODE_SOFTWARE_STANDBY
    );

    /* Normal execution will resume after switch SW1 is pressed */
    while(1);
}

static void SW1_handler(void)
{
}
```

Figure 10: Example of Software Standby Mode

5.8. Register Write Protection

The following example shows the use of Register Write Protection.

```
/* PDL functions and definitions */
#include "r_pdl_cgc.h"
#include "r_pdl_rwp.h"

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

void main(void)
{
    uint8_t prcr;
    uint8_t pwpr;

    /* Initialise the system clocks */
        NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.

    /* Turn off all register write protection */
    R_RWP_Control(
        PDL_RWP_ENABLE_CGC_WRITE |
        PDL_RWP_ENABLE_MODE_RESET_WRITE |
        PDL_RWP_ENABLE_LVD_WRITE |
        PDL_RWP_ENABLE_MPC_WRITE
    );

    /* Get status of protection */
    R_RWP_GetStatus(
        &prcr,
        &pwpr
    );

    /* Turn on all register write protection */
    R_RWP_Control(
        PDL_RWP_DISABLE_CGC_WRITE |
        PDL_RWP_DISABLE_MODE_RESET_WRITE |
        PDL_RWP_DISABLE_LVD_WRITE |
        PDL_RWP_DISABLE_MPC_WRITE
    );

    /* Get status of protection */
    R_RWP_GetStatus(
        &prcr,
        &pwpr
    );
}
```

Figure 11: Examples of Register Write Protection

5.9. DMA controller

The following example shows the use of triggers by software and IRQ pin edge detection.

Channel 0 will copy the string “Renesas RX220” into the destination area when a falling edge occurs on pin IRQ1 (P31).
Channel 1 will copy the string “Hello, World” into the destination area as soon as it is enabled.

```
/* PDL functions and definitions */
#include "r_pdl_dmac.h"
#include "r_pdl_intc.h"
#include "r_pdl_io_port.h"

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

/* Required for this example */
#include<string.h>

/* Callback function prototype */
void DMAC0_transfer_end_handler(void);

/* Data source and destination declarations */
const char source_string_1[]="Renesas RX220";
const char source_string_2[]="Hello, World";
volatile uint8_t destination_string_1[]=".....................";
volatile uint8_t destination_string_2[]=".....................";

void main(void)
{
    uint8_t StatusValue;
    uint32_t SourceAddr;
    uint32_t DestAddr;
    uint16_t TransferCount;
    uint16_t SizeCount;

    /* Set the CPU's Interrupt Priority Level to 0 */
    R_INTC_Write(
        PDL_INTC_REG_IPL,
        0
    );

    /* Enable control of LED1 */
    R_IO_PORT_Set(
        PDL_IO_PORT_1_5,
        PDL_IO_PORT_OUTPUT
    );

    /* Switch on LED1 */
    R_IO_PORT_Write(
        PDL_IO_PORT_1_5,
        0
    );

    /* Configure channel 0 */
    R_DMAC_Create(
        0,
        PDL_DMAC_BLOCK | PDL_DMAC_SOURCE_ADDRESS_PLUS | \
        PDL_DMAC_DESTINATION_ADDRESS_PLUS | \
        PDL_DMAC_SIZE_8 | PDL_DMAC_IRQ_END,
        PDL_DMAC_TRIGGER_IRQ1,
        source_string_1,
        destination_string_1,
        1,
        (uint16_t)strlen(source_string_1),
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
```

```
            DMAC0_transfer_end_handler,
            7
        );

        /* Configure channel 1 */
        R_DMAC_Create(
            1,
            PDL_DMAC_BLOCK | PDL_DMAC_SOURCE_ADDRESS_PLUS | \
            PDL_DMAC_DESTINATION_ADDRESS_PLUS | PDL_DMAC_SIZE_8,
            PDL_DMAC_TRIGGER_SW,
            source_string_2,
            destination_string_2,
            1,
            (uint16_t)strlen(source_string_2),
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_FUNC,
            0
        );

        /* Set IRQ1 pin to P31 */
        R_INTC_SetExtInterrupt(PDL_INTC_IRQ1_P31);

        /* Enable the IRQ1 interrupt */
        R_INTC_CreateExtInterrupt(
            PDL_INTC_IRQ1,
            PDL_INTC_FALLING | PDL_INTC_DMAC_TRIGGER_ENABLE,
            PDL_NO_FUNC,
            0
        );

        /* Enable channel 0 */
        R_DMAC_Control(
            0,
            PDL_DMAC_ENABLE,
            PDL_NO_PTR,
            PDL_NO_PTR,
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_DATA
        );

        /* Enable and start channel 1 */
        R_DMAC_Control(
            1,
            PDL_DMAC_ENABLE | PDL_DMAC_START,
            PDL_NO_PTR,
            PDL_NO_PTR,
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_DATA
        );

        /* Read the status for channel 0 */
        R_DMAC_GetStatus(
            0,
            &StatusValue,
            &SourceAddr,
            &DestAddr,
            &TransferCount,
            &SizeCount
        );
```

```
        while (1);
    }

    void DMAC0_transfer_end_handler(void)
    {
        /* Invert the port pin */
        R_IO_PORT_Modify(
            PDL_IO_PORT_1_5,
            PDL_IO_PORT_XOR,
            1
        );

        /* Stop all channels */
        R_DMAC_Control(
            0,
            PDL_DMAC_SUSPEND,
            PDL_NO_PTR,
            PDL_NO_PTR,
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_DATA
        );

        /* Shutdown channel 0 */
        R_DMAC_Destroy(
            0
        );
    }
```

Figure 12: Two examples of DMAC use

5.10. Data Transfer Controller

5.10.1. Block transfer mode

Figure 13 shows an example of Data Transfer Controller usage with a single block transfer.

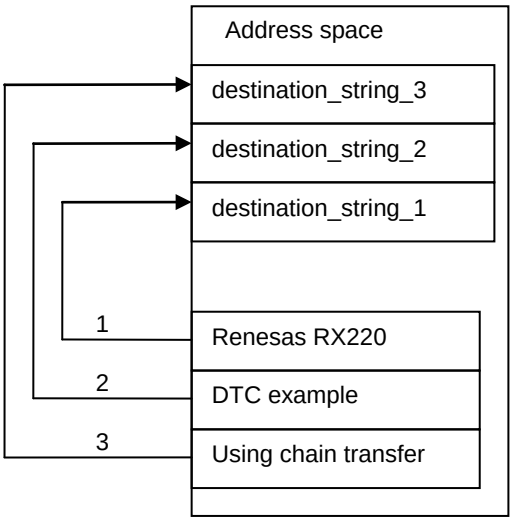
```
/* Peripheral driver function prototypes */
#include "r_pdl_dtc.h"
#include "r_pdl_io_port.h"
#include "r_pdl_intc.h"

/* RPDL 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 for the IRQ1-triggered transfer data area */
uint32_t dtc_irq1_transfer_data[4];

/* Data source and destination declarations */
const char source_string_1[]="Renesas RX220";
volatile uint8_t destination_string_1[]=".....................";

/* Callback function prototype */
void IRQ1_handler(void);

void main(void)
{
    /* Set the CPU's Interrupt Priority Level to 0 */
    R_INTC_Write(
        PDL_INTC_REG_IPL,
        0
    );

    /* Enable control of LED1 */
    R_IO_PORT_Set(
        PDL_IO_PORT_1_5,
        PDL_IO_PORT_OUTPUT
    );

    /* Set the DTC options */
    R_DTC_Set(
        PDL_NO_DATA,
        dtc_vector_table
    );

    /* Configure the DTC for IRQ1 */
    R_DTC_Create(
        PDL_DTC_BLOCK | PDL_DTC_DESTINATION | \
        PDL_DTC_SOURCE_ADDRESS_PLUS | PDL_DTC_DESTINATION_ADDRESS_PLUS | \
        PDL_DTC_SIZE_8 | \
        PDL_DTC_IRQ_COMPLETE | \
        PDL_DTC_TRIGGER_IRQ1,
        dtc_irq1_transfer_data,
        source_string_1,
        destination_string_1,
        1,
        (uint8_t)(strlen((char *)source_string_1))
    );

    /* Set IRQ1 pin to P31 */
    R_INTC_SetExtInterrupt(PDL_INTC_IRQ1_P31);
```

```
    /* Enable the SW1 (IRQ1) interrupt */
    R_INTC_CreateExtInterrupt(
        PDL_INTC_IRQ1,
        PDL_INTC_FALLING | PDL_INTC_DTC_TRIGGER_ENABLE,
        IRQ1_handler,
        7
    );

    /* Start the DTC */
    R_DTC_Control(
        PDL_DTC_START,
        PDL_NO_PTR,
        PDL_NO_PTR,
        PDL_NO_PTR,
        PDL_NO_DATA,
        PDL_NO_DATA
    );
    /* Wait for user key press of SW1 */

    while (1);
}

void IRQ1_handler(void)
{
    uint16_t StatusValue;
    uint32_t SourceAddr;
    uint32_t DestAddr;
    uint16_t TransferCount;

    /* Read the status and current source address for the IRQ1 transfer */
    R_DTC_GetStatus(
        dtc_irq1_transfer_data,
        &StatusValue,
        &SourceAddr,
        &DestAddr,
        &TransferCount,
        PDL_NO_DATA
    );

    /* Invert the port pin */
    R_IO_PORT_Modify(
        PDL_IO_PORT_1_5,
        PDL_IO_PORT_XOR,
        1
    );

    /* Re-enable IRQ1 as a DTC trigger */
    R_DTC_Control(
        PDL_DTC_TRIGGER_IRQ1,
        PDL_NO_PTR,
        PDL_NO_PTR,
        PDL_NO_PTR,
        PDL_NO_DATA,
        PDL_NO_DATA
    );
}
```

Figure 13: Example of DTC use

5.10.2. Chain transfer operation

Figure 14shows 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"

/* RPDL 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 RX220";
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)
{
    /* 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
        );

        /* Generate a software interrupt request */
        R_INTC_Write(PDL_INTC_REG_SWINTR,1);

        while(1);
    }
```

Figure 14: Example of DTC chain transfer

5.11. Port Output Enable

Figure 15 shows a usage example of Port Output Enable function.

```
/* PDL functions */
#include "r_pdl_poe.h"

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

void POE0_3_handler(void);
void POE8_handler(void);

void main(void)
{
    /* Configure POE pins */
    R_POE_Set(
        PDL_POE_0_MODE_EDGE   | PDL_POE_1_MODE_LOW_8 | \
        PDL_POE_2_MODE_LOW_16 | PDL_POE_3_MODE_LOW_128 | \
        PDL_POE_8_MODE_LOW_8,
        PDL_POE_0_PORT_D_7 | PDL_POE_1_PORT_D_6 | \
        PDL_POE_2_PORT_D_5 | PDL_POE_3_PORT_D_4 | \
        PDL_POE_8_PORT_D_3,
        PDL_POE_HI_Z_REQ_8_ENABLE | PDL_POE_HI_Z_REQ_OSTSTE | \
        PDL_POE_HI_Z_REQ_MTIOC0A | PDL_POE_HI_Z_REQ_MTIOC0B | \
        PDL_POE_HI_Z_REQ_MTIOC0C | PDL_POE_HI_Z_REQ_MTIOC0D
    );
    R_POE_Create(
        PDL_POE_IRQ_HI_Z_0_3_ENABLE | PDL_POE_IRQ_SHORT_3_4_ENABLE,
        POE0_3_handler,
        POE8_handler,
        15
    );

    while(1);
}

void POE0_3_handler(void)
{
    uint16_t StatusFlags;

    /* Read the POE status */
    R_POE_GetStatus(&StatusFlags);
    /* POE0 request? */
    if ((StatusFlags & BIT_0) != 0x0u)
    {
        /* Prevent further interrupts and try to clear the flag */
        R_POE_Control(
        PDL_NO_DATA,
        PDL_POE_FLAG_POE0_CLEAR,
        PDL_POE_IRQ_HI_Z_0_3_DISABLE
        );
    }
}

void POE8_handler(void)
{
    uint16_t StatusFlags;

    /* Read the POE status */
    R_POE_GetStatus(&StatusFlags);
    /* Prevent further interrupts and try to clear the flag */
    R_POE_Control(PDL_NO_DATA, PDL_POE_FLAG_POE8_CLEAR, PDL_POE_IRQ_HI_Z_8_DISABLE);
}
```

Figure 15: Example of Port Output Enable function

5.12. Event Link Controller

In this example the Event Link Controller links the 8-bit Timer with an I/O pin.

```
/* Peripheral driver function prototypes */
#include "r_pdl_cgc.h"
#include "r_pdl_tmr.h"
#include "r_pdl_io_port.h"
#include "r_pdl_elc.h"

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

void main(void)
{
    /* Initialise the system clocks */
    NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.

    /* Set pin B_0 as an output pin */
    R_IO_PORT_Set(PDL_IO_PORT_B_0,PDL_IO_PORT_OUTPUT);

    /* Configure TMR channel 0 */
    R_TMR_CreateChannel(
        0,
        PDL_TMR_CLK_PCLK_DIV_1 | PDL_TMR_CLEAR_CM_A,
        PDL_NO_DATA,
        0,
        199,
        99,
        PDL_NO_FUNC,
        PDL_NO_FUNC,
        PDL_NO_FUNC,
        0    );

    /* Enable the module, no ELC interrupts required */
    R_ELC_Create(
        PDL_NO_FUNC,
        PDL_NO_DATA
    );

    /* Create link between event 'TMR Channel 0' and module 'SinglePort 0' */
    R_ELC_Control(
        PDL_ELC_CREATE_LINK,
        PDL_ELC_LINK_MODULE_SINGLE_PORT_0,
        PDL_ELC_LINK_EVENT_TMR_CHANNEL_0_COMPARE_MATCH_A0
    );

    /* Configure 'SinglePort 0' as PB_0. Toggle output on event. */
    R_ELC_Control(
        PDL_ELC_SINGLE_PORT,
        PDL_ELC_SINGLE_PORT_0,
        PDL_ELC_PIN_PORT_B_0 | PDL_ELC_PIN_OUTPUT_TOGGLE
    );

    /* Enable All Links */
    R_ELC_Control(
        PDL_ELC_ENABLE,
        PDL_NO_DATA,
        PDL_NO_DATA
    );

    /* PB_0 will be toggling. */
    while(1){}
}
```

Figure 16: Example of Event Link Controller

5.13. Multi-Function Timer Pulse Unit

5.13.1. PWM mode 1

This example shows a usage of Multi-Function Timer Pulse Unit with PWM mode 1.

```
/* Peripheral driver function prototypes */
#include "r_pdl_mtu2.h"
#include "r_pdl_cgc.h"
#include "r_pdl_cmt.h"

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

void main(void)
{
    R_MTU2_Create_structure create_parameters;
    R_MTU2_ControlChannel_structure control_parameter;

    /* Prepare the main clock settings */
    R_CGC_Set(
        PDL_CGC_CLK_MAIN,
        PDL_CGC_MAIN_RESONATOR,
        20E6,
        20E6,
        20E6,
        20E6,
        20E6
    );

    /* Select the Main as the clock source */
    R_CGC_Control(
        PDL_CGC_CLK_MAIN,
        PDL_NO_DATA,
        PDL_NO_DATA
    );


    /* Allow 100us for the main clock to stabilise */
    R_CMT_CreateOneShot(
        0,
        PDL_NO_DATA,
        100E-6,
        PDL_NO_FUNC,
        0
    );

    /* Select the Main as the clock source */
    R_CGC_Control(
        PDL_CGC_CLK_MAIN,
        PDL_NO_DATA,
        PDL_NO_DATA
    );

    R_MTU2_Set(
        3,
        PDL_MTU2_PIN_3A_P14
    );

    /* Load the defaults */
    R_MTU2_Create_load_defaults(&create_parameters);

    create_parameters.channel_mode = PDL_MTU2_MODE_PWM1;
    create_parameters.counter_operation = PDL_MTU2_CLK_PCLK_DIV_256 |
PDL_MTU2_CLK_RISING | PDL_MTU2_CLEAR_TGRA;
    create_parameters.TGR_A_B_operation = PDL_MTU2_A_OC_HIGH | PDL_MTU2_B_OC_LOW;
    create_parameters.TGRA_TCNTV_value = 39062;
```

```
        create_parameters.TGRB_TCNTW_value = 33000;

        R_MTU2_Create(
            3,
            &create_parameters
        );

        control_parameter.control_setting = PDL_MTU2_START;
        control_parameter.register_selection = PDL_NO_DATA;

        R_MTU2_ControlChannel(
            3,
            &control_parameter
        );

        while(1);
    }
```

Figure 17: Example of MTU PWM mode 1

5.13.2. Reset-synchronized PWM mode

This example shows a usage of Multi-Function Timer Pulse Unit with Synchronized PWM mode.

```
/* Peripheral driver function prototypes */
#include "r_pdl_mtu2.h"
#include "r_pdl_cgc.h"
#include "r_pdl_cmt.h"

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

/* Global struct to avoid unwanted initial value */
R_MTU2_ControlUnit_structure control_unit_para;
R_MTU2_Create_structure create_parameters;
R_MTU2_ControlChannel_structure control_parameter;

void main(void)
{
    /* Initialise the system clocks */
    NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.

    /* Set channel 3 output pins MTIOC3B, MTIOC3D */
    R_MTU2_Set(
        3,
        PDL_MTU2_PIN_3B_P17 | PDL_MTU2_PIN_3D_P16
    );

    /* Set channel 4 output pins MTIOC4A, MTIOC4C, MTIOC4B, MTIOC4D */
    R_MTU2_Set(
        4,
        PDL_MTU2_PIN_4A_PE2 | PDL_MTU2_PIN_4C_PE1 | \
        PDL_MTU2_PIN_4B_PC2 | PDL_MTU2_PIN_4D_PC3
    );

    /* Load the defaults */
    R_MTU2_Create_load_defaults(&create_parameters);

    /* Set the non-default options */
    create_parameters.channel_mode = PDL_MTU2_MODE_NORMAL;
    create_parameters.TGRA_TCNTV_value = 30000;
    create_parameters.TGRB_TCNTW_value = 20000;

    R_MTU2_Create(
        4,
        &create_parameters
    );

    /* Load the defaults */
    R_MTU2_Create_load_defaults(&create_parameters);

    /* Set the non-default options */
    create_parameters.channel_mode = PDL_MTU2_MODE_PWM_RS;
    create_parameters.counter_operation = PDL_MTU2_CLK_PCLK_DIV_64 |
PDL_MTU2_CLEAR_TGRA;
    create_parameters.TGRA_TCNTV_value = 48000; //TGRA
    create_parameters.TGRB_TCNTW_value = 35000; //TGRB

    R_MTU2_Create(
        3,
        &create_parameters
    );

    control_unit_para.simultaneous_control = 0;
    control_unit_para.buffer_control = 0;
    control_unit_para.brushless_DC_motor_control = 0;
    control_unit_para.general_control = PDL_MTU2_PWM_RS_COMP_ENABLE;
```

```
        control_unit_para.register_selection = 0;
        control_unit_para.output_control = \
            PDL_MTU2_OUT_P_PHASE_3_ENABLE | PDL_MTU2_OUT_N_PHASE_3_ENABLE |\
            PDL_MTU2_OUT_P_PHASE_1_ENABLE | PDL_MTU2_OUT_N_PHASE_1_ENABLE | \
            PDL_MTU2_OUT_P_PHASE_2_ENABLE | PDL_MTU2_OUT_N_PHASE_2_ENABLE | \
            PDL_MTU2_OUT_P_PHASE_ALL_LOW_HIGH;

        R_MTU2_ControlUnit(
            0,
            &control_unit_para
        );

        control_parameter.control_setting = PDL_MTU2_START;
        control_parameter.register_selection = PDL_NO_DATA;

        R_MTU2_ControlChannel(
            3,
            &control_parameter
        );

        while(1);
    }
```

Figure 18: Example of MTU Reset-Synchornized PWM mode

5.14. 8-bit Timer

5.14.1. Periodic operation

Timer channel 1 is configured to provide pulses on pin TMO1, with a pulse width of 500ms and an on-time of 100ms.
Timer channel 2 is configured to provide pulses on pin TMO2, with a frequency of 1Hz and a duty cycle of 50%.

Notes: When running this example with RSK, the TMO1 and TMO2 pins are configured to connect to LED 3 and LED 2. The LEDs will be lighted up when the pin output low level.

```
/* Peripheral driver function prototypes */
#include "r_pdl_tmr.h"
#include "r_pdl_cgc.h"
#include "r_pdl_intc.h"

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

void main(void)
{
    /* Initialise the system clocks */
    NOTE: The code to initialise the system clock is omitted here.
    Please refer to 5.1 Clock Generation Circuit.

    /* Set the CPU's Interrupt Priority Level to 0 */
    R_INTC_Write(
        PDL_INTC_REG_IPL,
        0
    );

    /* Configure TMR1 input and output pins */
    R_TMR_Set(
        1,
        PDL_TMR_TMR1_TMO1_P17 | PDL_TMR_TMR1_TMCI1_P12 | PDL_TMR_TMR1_TMRI1_P24
    );

    /* Configure TMR1 input and output pins */
    R_TMR_Set(
        2,
        PDL_TMR_TMR2_TMO2_P16 | PDL_TMR_TMR2_TMCI2_PC6 | PDL_TMR_TMR2_TMRI2_PC5
    );

    /* Configure TMR1 for 500ms pulse width, 100ms on-time */
    R_TMR_CreatePeriodic(
        PDL_TMR_TMR1,
        PDL_TMR_PERIOD | PDL_TMR_OUTPUT_HIGH,
        500E-3,
        100E-3,
        PDL_NO_FUNC,
        PDL_NO_FUNC,
        0
    );

    /* Configure TMR2 for 1Hz frequency, 50% duty-cycle */
    R_TMR_CreatePeriodic(
        PDL_TMR_UNIT1,
        PDL_TMR_FREQUENCY | PDL_TMR_OUTPUT_HIGH,
        1E0,
        50,
        PDL_NO_FUNC,
        PDL_NO_FUNC,
        0
    );
}
```

Figure 19: Example of Pulse Output code

For full flexibility, the R_TMR_CreateChannel() function can be used.
In this example, Timer channel 0 is configured to provide pulses on pin TMO0, with a pulse width of 200 ticks of PCLKB and a duty cycle of 50%.
Note that the output transitions and counter clearing occur after the compare match has occurred. So the values for compare match A and compare match B should be 1 less than the required count.

```
/* Peripheral driver function prototypes */
#include "r_pdl_tmr.h"
#include "r_pdl_definitions.h"

void main(void)
{
    /* Initialise the system clocks */
    NOTE: The code to initialise the system clock is omitted here.
    Please refer to 5.1 Clock Generation Circuit.

    /* Configure TMR0 input and output pins */
R_TMR_Set(
        0,
        PDL_TMR_TMR0_TMO0_PB3
    );
    /* Configure TMR0 to clear on a compare match A, output 1 at a compare match A and
output 0 at a compare match B */
    R_TMR_CreateChannel(
        0,
        PDL_TMR_CLK_PCLK_DIV_1 | PDL_TMR_CLEAR_CM_A,
        PDL_TMR_OUTPUT_HIGH_CM_A | PDL_TMR_OUTPUT_LOW_CM_B,
        0,
        (200 - 1),
        (200 / 2) - 1,
        PDL_NO_FUNC,
        PDL_NO_FUNC,
        PDL_NO_FUNC,
        0
    );
}
```

Figure 20: Example of Pulse Output code

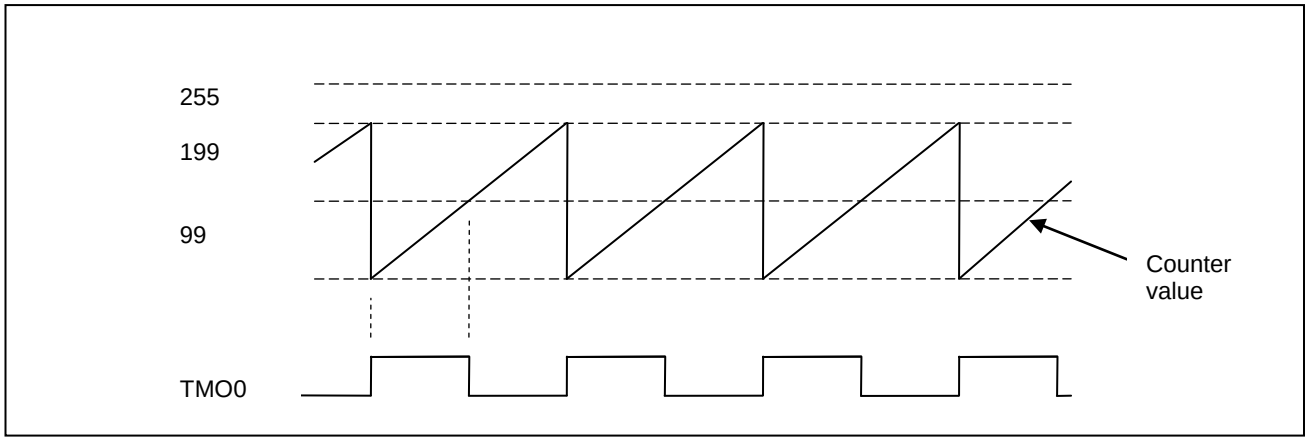


Figure 21: Example of pulse output operation

5.15. Compare Match Timer

Figure 22 shows an example of Compare Match Timer usage. One channel is used to generate interrupts at regular intervals.

Channel 0 is used to generate interrupt using frequency configuration at 1 kHz then change to 5 Hz. When channel 0 is created, it will not start until 1 second delay passed.
Channel 1 is used to generate interrupt using period of 0.1 second. Channel 1 starts right after it is created and stop when 1 second delay passed.
Both channel 0 and channel 1 will be destroyed before the main loop.
Channel 2 and channel 3 are used to create one-shot delay.

```
    /* PDL functions */
    #include "r_pdl_cgc.h"
    #include "r_pdl_cmt.h"
    #include "r_pdl_intc.h"
    #include "r_pdl_io_port.h"

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

    void CMT0_handler(void);
    void CMT1_handler(void);

    void main(void)
    {
        uint8_t flags;
        uint16_t counter;

        /* Configure main clock operation using a 20.0 MHz crystal */
        /* ICLK = 20 MHz, PCLKD = 20 MHz, PCLKB = 20 MHz , FCLK = 20 MHz */
        R_CGC_Set(
            PDL_CGC_CLK_MAIN,
            PDL_CGC_MAIN_RESONATOR,
            20E6,
            20E6,
            20E6,
            20E6,
            20E6
        );

        /* Configure HOCO operation */
        /* ICLK = 32 MHz, PCLKD = 32 MHz, PCLKB = 32 MHz , FCLK = 32 MHz */
        R_CGC_Set(
            PDL_CGC_CLK_HOCO,
            PDL_CGC_HOCO_32000,
            32E6,
            32E6,
            32E6,
            32E6,
            32E6
        );

        /* Wait 100us for the main clock to stabilise */
        R_CMT_CreateOneShot(3, PDL_NO_DATA, 100E-6, PDL_NO_FUNC, 0);

        /* Select HOCO as the clock source */
        R_CGC_Control(PDL_CGC_CLK_HOCO, PDL_NO_DATA, PDL_NO_DATA);

        /* Set the CPU's Interrupt Priority Level to 0 */
        R_INTC_Write(PDL_INTC_REG_IPL, 0);

        /* Configure port pins for output */
        R_IO_PORT_Set(PDL_IO_PORT_1_4, PDL_IO_PORT_OUTPUT);
        R_IO_PORT_Set(PDL_IO_PORT_1_5, PDL_IO_PORT_OUTPUT);

        /* Turn on LED0 and LED1 */
```

```
        R_IO_PORT_Write(PDL_IO_PORT_1_4, 0);
        R_IO_PORT_Write(PDL_IO_PORT_1_5, 0);

        /* Configure CMT channel 0 for 1kHz operation, but not start CMT first */
        R_CMT_Create(
            0,
            PDL_CMT_FREQUENCY | PDL_CMT_STOP,
            1E3,
            CMT0_handler,
            7
        );

        /* Configure CMT channel 1 at 0.1 second period and start CMT */
        R_CMT_Create(
            1,
            PDL_CMT_PERIOD,
            1E-1,
            CMT1_handler,
            7
        );

        /* Change CMT channel 0 configuration frequency to 5Hz */
        R_CMT_Control(0, PDL_CMT_FREQUENCY, 5E0);

        /* Read flag and count value */
        R_CMT_Read(0, PDL_NO_PTR, PDL_NO_PTR);
        R_CMT_Read(1, &flags, &counter);

        /* Wait for 1s */
        R_CMT_CreateOneShot(2, PDL_NO_DATA, 1E0, PDL_NO_FUNC, 0);

        /* Start CMT channel 0 */
        R_CMT_Control(0, PDL_CMT_START, PDL_NO_DATA);
        /* Stop CMT channel 1 */
        R_CMT_Control(1, PDL_CMT_STOP, PDL_NO_DATA);

        /* Wait for 1s */
        R_CMT_CreateOneShot(2, PDL_NO_DATA, 1E0, PDL_NO_FUNC, 0);

        /* Destroy CMT unit 0 (channel 0 and 1) */
        R_CMT_Destroy(0);

        /* Main loop */
        while (1)
        {

        }
    }

    void CMT0_handler(void)
    {
        /* Toggle LED0 */
        R_IO_PORT_Modify(PDL_IO_PORT_1_4, PDL_IO_PORT_XOR, 1);
    }

    void CMT1_handler(void)
    {
        /* Toggle LED1 */
        R_IO_PORT_Modify(PDL_IO_PORT_1_5, PDL_IO_PORT_XOR, 1);
    }
```

Figure 22: Example of Compare Match Timer use

5.16. Real-time Clock

5.16.1. Use case of RTC (configuration and use case)

These examples show initialization procedure of simple RTC use case

1) Configuration CGC and RTC counting by sub-clock (only RTC count source) in calendar count mode

Figure 23 shows an example of sub clock used as count source and main clock used as system clock before using the Real-time clock in Calendar count mode.

```
/* PDL functions */
#include "r_pdl_cgc.h"
#include "r_pdl_cmt.h"
#include "r_pdl_rtc.h"

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

void main(void)
{
    volatile uint32_t date, time;

    /* Prepare the main clock settings */
    /* ICLK = 20 MHz, PCLKD = 20 MHz, PCLKB = 20 MHz, FCLK = 20 MHz */
    R_CGC_Set(
        PDL_CGC_CLK_MAIN,
        PDL_CGC_MAIN_RESONATOR,
        20E6,
        20E6,
        20E6,
        20E6,
        20E6,
        PDL_NO_DATA
    );

    /* Configure the sub clock settings */
    R_CGC_Set(
        PDL_CGC_CLK_SUB_CLOCK,
        PDL_CGC_SUB_CLOCK_CL_STANDARD,
        32768,
        32768,
        32768,
        32768,
        32768,
        PDL_CGC_SUB_2
    );

    /* Generate the 2s delay before enabling RTC by CGC_Control */
    R_CMT_CreateOneShot(
        0,
        PDL_NO_DATA,
        2,
        PDL_NO_FUNC,
        0
    );

    /* Select RTC to be used */
    R_CGC_Control(
        PDL_CGC_CLK_MAIN,
        PDL_NO_DATA,
        PDL_CGC_RTC_TO_BE_USED
    );

    /* If Cold-start is detected the RTC clock should be re-started */
    R_RTC_Create(
```

```
            PDL_NO_DATA,
            0x04110710, /* WED 11:07:10 */
            0x20140116, /* 20140116 */
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_FUNC,
            PDL_NO_DATA,
            PDL_NO_FUNC,
            PDL_NO_DATA
        );

        while(1)
        {
            R_RTC_Read(
                PDL_RTC_READ_CURRENT,
                PDL_NO_PTR,
                &time,
                &date
            );
        }
    }
```

Figure 23: Example of configuration CGC and RTC counting by sub clock (Only RTC count source) in Calendar count mode

2) Configuration CGC and RTC counting by sub-clock (only RTC count source) in binary count mode

Figure 24 shows an example of sub clock used as count source and main clock used as system clock before using the Real-time clock in Binary count mode.

```
/* PDL functions */
#include "r_pdl_cgc.h"
#include "r_pdl_cmt.h"
#include "r_pdl_rtc.h"

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

void main(void)
{
    volatile uint8_t flags;
    volatile uint32_t count, alarm, r_mask;

    /* Prepare the main clock settings */
    /* ICLK = 20 MHz, PCLKD = 20 MHz, PCLKB = 20 MHz, FCLK = 20 MHz */
    R_CGC_Set(
        PDL_CGC_CLK_MAIN,
        PDL_CGC_MAIN_RESONATOR,
        20E6,
        20E6,
        20E6,
        20E6,
        20E6,
        PDL_NO_DATA
    );

    /* Prepare the Sub-clock settings */
    R_CGC_Set(
        PDL_CGC_CLK_SUB_CLOCK,
        PDL_CGC_SUB_CLOCK_CL_STANDARD,
        32768,
        32768,
        PDL_NO_DATA,
        32768, /* PCLKB clock = sub-clock when sub-clock is source clock */
        32768,
        PDL_CGC_SUB_32768
    );

    /* Generate the 2s delay before enabling RTC by CGC_Control */
    R_CMT_CreateOneShot(
        0,
        PDL_NO_DATA,
        2,
        PDL_NO_FUNC,
        0
    );

    /* Select RTC to be used */
    R_CGC_Control(
        PDL_CGC_CLK_MAIN,
        PDL_NO_DATA,
        PDL_CGC_RTC_TO_BE_USED
    );

    /* If Cold-start is detected the RTC clock should be re-started */
    R_RTC_CreateBinary(
        PDL_NO_DATA,  /* Configuration */
        0x12345678,   /* Current_count  */
        PDL_NO_DATA,  /* Periodic */
        PDL_NO_DATA,  /* Alarm_count */
        PDL_NO_DATA,  /* Alarm_mask */
        PDL_NO_FUNC,  /* Alarm handler */
```

```
            PDL_NO_DATA,  /* Alarm priority */
            PDL_NO_FUNC,  /* Periodic Handler */
            PDL_NO_DATA   /* Periodic priority */
        );

        while(1)
        {
            R_RTC_ReadBinary(
                &flags,
                &count,
                &alarm,
                &r_mask);
        }
    }
```

Figure 24: Example of configuration CGC and RTC counting by sub clock (Only RTC count source) in Binary count mode

3) Configuration CGC and RTC counting by sub-clock (both RTC count source and System clock) in calendar count mode

Figure 25 shows an example of sub clock used as both RTC count source and system clock in Calendar count mode.

```
/* PDL functions */
#include "r_pdl_cgc.h"
#include "r_pdl_cmt.h"
#include "r_pdl_rtc.h"

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

void main(void)
{
    volatile uint32_t date,time;

    /* Prepare the LOCO settings */
    R_CGC_Set(
        PDL_CGC_CLK_LOCO,
        PDL_NO_DATA,
        125E3,
        125E3,
        PDL_NO_DATA,
        125E3,
        125E3,
        PDL_NO_DATA
    );

    /* Prepare the Sub-clock settings */
    R_CGC_Set(
        PDL_CGC_CLK_SUB_CLOCK,
        PDL_CGC_SUB_CLOCK_CL_STANDARD,
        32768,
        32768,
        PDL_NO_DATA,
        32768, /* PCLKB clock = sub-clock when sub-clock is source clock */
        32768,
        PDL_CGC_SUB_32768
    );

    /* Generate the 2s delay before enabling RTC by CGC_Control */
    R_CMT_CreateOneShot(
        0,
        PDL_NO_DATA,
        2.0,
        PDL_NO_FUNC,
        0
    );

    /* Select the HOCO as the clock source */
    R_CGC_Control(
        PDL_CGC_CLK_SUB_CLOCK,
        PDL_NO_DATA,
        PDL_CGC_RTC_TO_BE_USED
    );

    /* If Cold-start is detected the RTC clock should be re-started */
    R_RTC_Create(
        PDL_NO_DATA,
        0x04110710, /* WED 11:07:10 */
        0x20140116, /* 20140116 */
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_FUNC,
        PDL_NO_DATA,
        PDL_NO_FUNC,
```

```
            PDL_NO_DATA
        );

        while(1)
        {
            R_RTC_Read(
                PDL_RTC_READ_CURRENT,
                PDL_NO_PTR,
                &time,
                &date
            );
        }
    }
```

Figure 25: Example of configuration CGC and RTC counting by sub clock (Both RTC count source and System clock) in Calendar count mode

4) Configuration CGC and RTC counting by sub-clock (both RTC count source and System clock) in binary count mode

Figure 26 shows an example of sub clock used as both RTC count source and system clock in Binary count mode.

```
/* PDL functions */
#include "r_pdl_cgc.h"
#include "r_pdl_cmt.h"
#include "r_pdl_rtc.h"

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

void main(void)
{
    volatile uint8_t flags;
    volatile uint32_t count, alarm, r_mask;

    /* Prepare the LOCO settings */
    R_CGC_Set(
        PDL_CGC_CLK_LOCO,
        PDL_NO_DATA,
        125E3,
        125E3,
        PDL_NO_DATA,
        125E3,
        125E3,
        PDL_NO_DATA
    );

    /* Prepare the Sub-clock settings */
    R_CGC_Set(
        PDL_CGC_CLK_SUB_CLOCK,
        PDL_CGC_SUB_CLOCK_CL_STANDARD,
        32768,
        32768,
        PDL_NO_DATA,
        32768, /* PCLKB clock = sub-clock when sub-clock is source clock */
        32768,
        PDL_CGC_SUB_32768
    );

    /* Generate the 2s delay before enabling RTC by CGC_Control */
    R_CMT_CreateOneShot(
        0,
        PDL_NO_DATA,
        2.0,
        PDL_NO_FUNC,
        0
    );

    /* Select the HOCO as the clock source */
    R_CGC_Control(
        PDL_CGC_CLK_SUB_CLOCK,
        PDL_NO_DATA,
        PDL_CGC_RTC_TO_BE_USED
    );

    /* If Cold-start is detected the RTC clock should be re-started */
    R_RTC_CreateBinary(
        PDL_NO_DATA,  /* Configuration */
        0x12345678,   /* Current_count  */
        PDL_NO_DATA,  /* Periodic */
        PDL_NO_DATA,  /* Alarm_count */
        PDL_NO_DATA,  /* Alarm_mask */
        PDL_NO_FUNC,  /* Alarm handler */
        PDL_NO_DATA,  /* Alarm priority */
        PDL_NO_FUNC,  /* Periodic Handler */
```

```
            PDL_NO_DATA   /* Periodic priority */
        );

        while(1)
        {
            R_RTC_ReadBinary(
                &flags,
                &count,
                &alarm,
                &r_mask);
        }
    }
```

Figure 26: Example of configuration CGC and RTC counting by sub clock (Both RTC count source and System clock) in Binary count mode

5.16.2. Initialization in case of RTC is not used

1) Initialize RTC with providing sub-clock (use-case sub clock is available)

Figure 27 shows an example of initialization in case of RTC is not used and sub clock is available.

```
/* PDL functions */
#include "r_pdl_cgc.h"
#include "r_pdl_cmt.h"

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

void main(void)
{
    /* Prepare the LOCO settings */
    R_CGC_Set(
        PDL_CGC_CLK_LOCO,
        PDL_NO_DATA,
        125E3,
        125E3,
        PDL_NO_DATA,
        125E3,
        125E3,
        PDL_NO_DATA
    );

    /* Configure the HOCO settings */
    R_CGC_Set(
        PDL_CGC_CLK_HOCO,
        PDL_CGC_HOCO_50000,
        50E6,
        25E6,
        PDL_NO_DATA,
        25E6,
        25E6,
        PDL_NO_DATA
    );

    /* Prepare the Sub-clock settings */
    R_CGC_Set(
        PDL_CGC_CLK_SUB_CLOCK,
        PDL_CGC_SUB_CLOCK_CL_STANDARD,
        32768,
        32768,
        PDL_NO_DATA,
        32768, /* PCLKB clock = sub-clock when sub-clock is source clock */
        32768,
        PDL_CGC_SUB_32768
    );

    /* Generate the 2s delay before enabling RTC by CGC_Control */
    R_CMT_CreateOneShot(
        0,
        PDL_NO_DATA,
        2.0,
        PDL_NO_FUNC,
        0
    );

    /* Select the HOCO as the clock source */
    R_CGC_Control(
        PDL_CGC_CLK_HOCO,
        PDL_NO_DATA,
        PDL_CGC_RTC_NOT_USE
    );
```

```
        while (1);
    }
```

Figure 27: Example of initialization of not using RTC with available sub clock

2) Initialize RTC without providing clock

Figure 28 shows an example of initialization in case of RTC is not used and RTC count source is not defined.

```
/* PDL functions */
#include "r_pdl_cgc.h"
#include "r_pdl_cmt.h"

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

void main(void)
{
    /* Prepare the LOCO settings */
    R_CGC_Set(
        PDL_CGC_CLK_LOCO,
        PDL_NO_DATA,
        125E3,
        125E3,
        PDL_NO_DATA,
        125E3,
        125E3,
        PDL_NO_DATA
    );

    /* Configure the HOCO settings */
    R_CGC_Set(
        PDL_CGC_CLK_HOCO,
        PDL_CGC_HOCO_50000,
        50E6,
        25E6,
        PDL_NO_DATA,
        25E6,
        25E6,
        PDL_NO_DATA
    );

    /* Generate the 2s delay before disabling RTC by CGC_Control */
    R_CMT_CreateOneShot(
        0,
        PDL_NO_DATA,
        2.0,
        PDL_NO_FUNC,
        0
    );

    /* Select the HOCO as the clock source */
    R_CGC_Control(
        PDL_CGC_CLK_HOCO,
        PDL_NO_DATA,
        PDL_CGC_SUB_CLOCK_DISABLE | PDL_CGC_RTC_NOT_USE
    );

    while (1);
}
```

Figure 28: Example of initialization of not using RTC without providing clock

5.16.3. Use case of RTC over reset and power comsumption

1) Wake up from sleep mode

Figure 29 shows an example of using the Real-time Clock wake up from sleep mode. The sub clock is used as RTC count source and HOCO is set as system clock.

```
#include <stdio.h>
#include <string.h>

/* PDL functions */
#include "r_pdl_cgc.h"
#include "r_pdl_lpc.h"
#include "r_pdl_cmt.h"
#include "r_pdl_mcu.h"
#include "r_pdl_rtc.h"
#include "r_pdl_sci.h"

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

#define RSK_SCI_CHANNEL 1

static void SetClocks(void);
static void Alarm_handler(void);

volatile bool bEnterSleepMode = false;

void main(void)
{
    uint8_t flags;
    uint32_t time;
    uint32_t date;
    uint32_t time_previous = 0;
    uint8_t buffer[50];
    uint16_t status=0;

    /* Sets system clock */
    SetClocks();

    /* Create async for debug output */
    R_SCI_Set(
        RSK_SCI_CHANNEL,
        PDL_SCI_PIN_SCI1_RXD1_P30 | \
        PDL_SCI_PIN_SCI1_TXD1_P26
    );

    R_SCI_Create(RSK_SCI_CHANNEL,
                 PDL_SCI_8N1 | PDL_SCI_ASYNC,
                 9600,
                 1);

    /* Check warm/cold start flag (CWSF = 1):
    call R_RTC_CreateWarm to start up the RTC if warm/cold start flag is detected
    (power ON from warm start) */

    /* Get Reset Status Flag */
    R_MCU_GetStatus(
        PDL_NO_PTR,
        &status,
        PDL_NO_PTR,
        PDL_NO_PTR
    );

    if (((status & BIT_8) == (uint16_t)(BIT_8))&& ((RTC.RCR2.BIT.START) != 0))
    {
        /* If warm-start is detected and RTC is running, then warm start */
```

```
            R_SCI_Send(RSK_SCI_CHANNEL, PDL_NO_DATA,
             "\r\nRTC Start in Warm start mode: Control to change time\r\n", 0, PDL_NO_FUNC);

            /* Warm wake up : Read time */
            R_RTC_Read(PDL_RTC_READ_CURRENT, &flags, &time, &date);

            sprintf((char*)buffer, "RTC Time before changing = %d%d:%d%d:%d%d \r\n",
                    (int)(time & 0xF00000) >> 20,
                    (int)(time & 0x0F0000) >> 16,
                    (int)(time & 0x00F000) >> 12,
                    (int)(time & 0x000F00) >> 8,
                    (int)(time & 0x0000F0) >> 4,
                    (int)(time & 0x00000F) >> 0);

            R_SCI_Send(RSK_SCI_CHANNEL, PDL_NO_DATA, buffer, 0, PDL_NO_FUNC);

            R_RTC_CreateWarm(
                Alarm_handler, /* Alarm handler */
                15,            /* Alarm priority */
                PDL_NO_FUNC,   /* Periodic Handler */
                PDL_NO_DATA    /* Periodic priority */
            );

            R_RTC_Read(
                PDL_RTC_READ_CURRENT,
                PDL_NO_PTR,
                &time,
                &date
            );

            /* Configure the clock */
            R_RTC_Control(
                PDL_NO_DATA,
                PDL_RTC_UPDATE_ALARM_TIME,
                PDL_NO_DATA,
                PDL_NO_DATA,
                (time + 0x10),  /* Alarm in another 10 seconds */
                PDL_NO_DATA,
                PDL_NO_DATA, /* Error Adjust */
                PDL_NO_DATA  /* Periodic */
            );
        }
        else
        {
            /* If Cold-start is detected the RTC clock should be re-started */

            R_SCI_Send(RSK_SCI_CHANNEL, PDL_NO_DATA,
             "\r\nRTC Start in Cold start mode: Initailize RTC\r\n", 0, PDL_NO_FUNC);

            if (R_RTC_Create(
                PDL_RTC_ALARM_TIME_ENABLE | PDL_RTC_ALARM_DATE_ENABLE,
                0xFF114250,  /* Automatic day of week, 11:42:50  */
                0x20131118,  /* 18-Nov-2013  */
                PDL_NO_DATA, /* Periodic */
                0xFF114300,  /* Alarm in 10 seconds  */
                0x20131118,  /* 18-Nov-2013  */
                Alarm_handler,
                15,
                PDL_NO_FUNC,
                PDL_NO_DATA
            ) == false)
            {
                R_SCI_Send(RSK_SCI_CHANNEL, PDL_NO_DATA,
                "\r\nRTC_Create error in Cold start mode\r\n", 0, PDL_NO_FUNC);
                while(1);
            }

            /* After the complete initialization, Set the warm start indicator */
```

```
            R_MCU_Control(
                PDL_MCU_WARM_START
            );
        }

        /* This call should cancel the settings made in above call to R_LPC_Create */
        R_LPC_Create(
            PDL_LPC_MIDDLE_SPEED_MODE_B,
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_DATA
        );

        R_RTC_Read(
            PDL_RTC_READ_CURRENT,
            PDL_NO_PTR,
            &time,
            &date
        );

        /* Enter sleep mode after alarm in 10sec */
        while (bEnterSleepMode == false);

        /* Enter sleep mode.  An internal reset will occur. */
        if(false == R_LPC_Control(PDL_LPC_MODE_SLEEP))
        {
            while(1);
        }

        /* It is correct to be here if have just woken from sleep mode. */
        while(1)
        {
            /* Read time */
            R_RTC_Read(PDL_RTC_READ_CURRENT, &flags, &time, &date);

            /* If no carry error output the time */
            if (0 == (flags & BIT_6))
            {
                /* Has time changed */
                if ((time & 0xFFFFFF) != (time_previous & 0xFFFFFF))
                {
                    time_previous = time;
                    sprintf((char*)buffer, "Time = %d%d:%d%d:%d%d \r\n",
                                    (int)(time & 0xF00000) >> 20,
                                    (int)(time & 0x0F0000) >> 16,
                                    (int)(time & 0x00F000) >> 12,
                                    (int)(time & 0x000F00) >> 8,
                                    (int)(time & 0x0000F0) >> 4,
                                    (int)(time & 0x00000F) >> 0);

                    R_SCI_Send(
                           RSK_SCI_CHANNEL,
                           PDL_NO_DATA,
                           buffer,
                           0,
                           PDL_NO_FUNC);
                }
            }
        }

    }


    static void SetClocks(void)
    {
        /* Prepare the LOCO settings */
        R_CGC_Set(
            PDL_CGC_CLK_LOCO,
```

```
            PDL_NO_DATA,
            125E3,
            125E3,
            PDL_NO_DATA,
            125E3,
            125E3,
            PDL_NO_DATA
        );

        /* Configure the HOCO settings */
        R_CGC_Set(
            PDL_CGC_CLK_HOCO,
            PDL_CGC_HOCO_50000,
            50E6,
            25E6,
            PDL_NO_DATA,
            25E6,
            25E6,
            PDL_NO_DATA
        );

        /* Prepare the Sub-clock settings */
        R_CGC_Set(
            PDL_CGC_CLK_SUB_CLOCK,
            PDL_CGC_SUB_CLOCK_CL_STANDARD,
            32768,
            32768,
            PDL_NO_DATA,
            32768, /* PCLKB clock = sub-clock when sub-clock is source clock */
            32768,
            PDL_CGC_SUB_32768
        );

        /* Wait for the Subclock stabilisation time (2 seconds minimum) */
        /* NOTE: As curently running from the Sub-clock the R_CMT_CreateOneShot
        max time limit is > 2 Secs. */
        R_CMT_CreateOneShot(
            0,
            PDL_NO_DATA,
            2.0,
            PDL_NO_FUNC,
            0
        );

        /* Select the HOCO as the clock source */
        R_CGC_Control(
            PDL_CGC_CLK_HOCO,
            PDL_NO_DATA,
            PDL_CGC_RTC_TO_BE_USED
        );
    }

    void Alarm_handler(void)
    {
        uint8_t flags;
        uint32_t time;
        uint32_t date;
        uint8_t buffer[50];

        /* Read time */
        R_RTC_Read(PDL_RTC_READ_CURRENT, &flags, &time, &date);

        sprintf((char*)buffer, "Time = %d%d:%d%d:%d%d \r\n",
                (int)(time & 0xF00000) >> 20,
                (int)(time & 0x0F0000) >> 16,
                (int)(time & 0x00F000) >> 12,
                (int)(time & 0x000F00) >> 8,
                (int)(time & 0x0000F0) >> 4,
```

```
                (int)(time & 0x00000F) >> 0);

        R_SCI_Send(RSK_SCI_CHANNEL, PDL_NO_DATA, buffer, 0, PDL_NO_FUNC);

        /* Configure the clock */
        R_RTC_Control(
            PDL_NO_DATA,
            PDL_RTC_UPDATE_ALARM_TIME,
            PDL_NO_DATA,
            PDL_NO_DATA,
            (time + 0x10),  /* Alarm in another 10 seconds */
            PDL_NO_DATA,
            PDL_NO_DATA, /* Error Adjust */
            PDL_NO_DATA  /* Periodic */
        );

        /* Enter into sleep mode? */
        If (true != bEnterSleepMode)
        {
            bEnterSleepMode = true;
        }

        nop();

    }
```

Figure 29: Example of using RTC is used and wake up from sleep mode

2) Wake up from software standby mode

Figure 30 shows an example of using the RTC wake up from software standby mode. The HOCO is used as system and and sub-clock as RTC count source.

```
#include <stdio.h>
#include <string.h>

/* PDL functions */
#include "r_pdl_cgc.h"
#include "r_pdl_lpc.h"
#include "r_pdl_cmt.h"
#include "r_pdl_mcu.h"
#include "r_pdl_rtc.h"
#include "r_pdl_sci.h"

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

#define RSK_SCI_CHANNEL 1

static void SetClocks(void);
static void Alarm_handler(void);

volatile bool bSoftStdbyEnter = false;

void main(void)
{
    uint16_t status=0;
    uint8_t flags;
    uint32_t time;
    uint32_t date;
    uint8_t buffer[50];
    uint32_t time_previous = 0;

    /* Sets system clock */
    SetClocks();

    /* Create async for debug output */
    R_SCI_Set(
        RSK_SCI_CHANNEL,
        PDL_SCI_PIN_SCI1_RXD1_P30 | \
        PDL_SCI_PIN_SCI1_TXD1_P26
    );

    R_SCI_Create(RSK_SCI_CHANNEL,
                PDL_SCI_8N1 | PDL_SCI_ASYNC,
                9600,
                1);

    /* Check warm/cold start flag (CWSF = 1):
    call R_RTC_CreateWarm to start up the RTC if warm/cold start flag is detected
    (power ON from warm start) */

    /* Get Reset Status Flag */
    R_MCU_GetStatus(
        PDL_NO_PTR,
        &status,
        PDL_NO_PTR,
        PDL_NO_PTR
    );

    if (((status & BIT_8) == (uint16_t)(BIT_8))&& ((RTC.RCR2.BIT.START) != 0))
    {
        /* If warm-start is detected and RTC is running, then warm start */
        R_SCI_Send(RSK_SCI_CHANNEL, PDL_NO_DATA,
         "\r\nRTC Start in Warm start mode: Control to change time\r\n", 0, PDL_NO_FUNC);
```

```
            /* Warm wake up : Read time */
            R_RTC_Read(PDL_RTC_READ_CURRENT, &flags, &time, &date);

            sprintf((char*)buffer, "RTC Time before changing = %d%d:%d%d:%d%d \r\n",
                    (int)(time & 0xF00000) >> 20,
                    (int)(time & 0x0F0000) >> 16,
                    (int)(time & 0x00F000) >> 12,
                    (int)(time & 0x000F00) >> 8,
                    (int)(time & 0x0000F0) >> 4,
                    (int)(time & 0x00000F) >> 0);

            R_SCI_Send(RSK_SCI_CHANNEL, PDL_NO_DATA, buffer, 0, PDL_NO_FUNC);

            R_RTC_CreateWarm(
                Alarm_handler, /* Alarm handler */
                15,            /* Alarm priority */
                PDL_NO_FUNC,   /* Periodic Handler */
                PDL_NO_DATA    /* Periodic priority */
            );

            R_RTC_Read(
                PDL_RTC_READ_CURRENT,
                PDL_NO_PTR,
                &time,
                &date
            );

            /* Configure the clock */
            R_RTC_Control(
                PDL_NO_DATA,
                PDL_RTC_UPDATE_ALARM_TIME,
                PDL_NO_DATA,
                PDL_NO_DATA,
                (time + 0x10),  /* Alarm in another 10 seconds */
                PDL_NO_DATA,
                PDL_NO_DATA, /* Error Adjust */
                PDL_NO_DATA  /* Periodic */
            );
        }
        else
        {
            /* If Cold-start is detected the RTC clock should be re-started */
            R_SCI_Send(RSK_SCI_CHANNEL, PDL_NO_DATA,
             "\r\nRTC Start in Cold start mode: Initialize RTC\r\n", 0, PDL_NO_FUNC);

            if (R_RTC_Create(
                PDL_RTC_ALARM_TIME_ENABLE | PDL_RTC_ALARM_DATE_ENABLE,
                0xFF114250,  /* Automatic day of week, 11:42:50  */
                0x20131118,  /* 18-Nov-2013  */
                PDL_NO_DATA, /* Periodic */
                0xFF114300,  /* Alarm in 10 seconds  */
                0x20131118,  /* 18-Nov-2013  */
                Alarm_handler,
                15,
                PDL_NO_FUNC,
                PDL_NO_DATA
            ) == false)
            {
                R_SCI_Send(RSK_SCI_CHANNEL, PDL_NO_DATA,
                "\r\nRTC_Create error in Cold start mode\r\n", 0, PDL_NO_FUNC);
                while(1);
            }

            /* After the complete initialization, Set the warm start indicator */
            R_MCU_Control(
                PDL_MCU_WARM_START
            );
        }
```

```
    /* This call should cancel the settings made in above call to R_LPC_Create */
    R_LPC_Create(
        PDL_LPC_MIDDLE_SPEED_MODE_A,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA
    );

    R_RTC_Read(
        PDL_RTC_READ_CURRENT,
        PDL_NO_PTR,
        &time,
        &date
    );

    /* Enter software standby mode after alarm in 10sec */
    while (bSoftStdbyEnter == false);

    /* Enter software standby mode.  An internal reset will occur. */
    if(false == R_LPC_Control(PDL_LPC_MODE_SOFTWARE_STANDBY))
    {
        while(1);
    }

    while(1)
    {
        /* Read time */
        R_RTC_Read(PDL_RTC_READ_CURRENT, &flags, &time, &date);

        /* If no carry error output the time */
        if (0 == (flags & BIT_6))
        {
            /* Has time changed */
            if ((time & 0xFFFFFF) != (time_previous & 0xFFFFFF))
            {
                time_previous = time;
                sprintf((char*)buffer, "Time = %d%d:%d%d:%d%d \r\n",
                                (int)(time & 0xF00000) >> 20,
                                (int)(time & 0x0F0000) >> 16,
                                (int)(time & 0x00F000) >> 12,
                                (int)(time & 0x000F00) >> 8,
                                (int)(time & 0x0000F0) >> 4,
                                (int)(time & 0x00000F) >> 0);

                R_SCI_Send(
                        RSK_SCI_CHANNEL,
                        PDL_NO_DATA,
                        buffer,
                        0,
                        PDL_NO_FUNC);
            }
        }
    }
}


static void SetClocks(void)
{
    /* Prepare the LOCO settings */
    R_CGC_Set(
        PDL_CGC_CLK_LOCO,
        PDL_NO_DATA,
        125E3,
        125E3,
        PDL_NO_DATA,
        125E3,
        125E3,
```

```
            PDL_NO_DATA
        );

        /* Configure the HOCO settings */
        R_CGC_Set(
            PDL_CGC_CLK_HOCO,
            PDL_CGC_HOCO_50000,
            50E6,
            25E6,
            PDL_NO_DATA,
            25E6,
            25E6,
            PDL_NO_DATA
        );

        /* Prepare the Sub-clock settings */
        R_CGC_Set(
            PDL_CGC_CLK_SUB_CLOCK,
            PDL_CGC_SUB_CLOCK_CL_STANDARD,
            32768,
            32768,
            PDL_NO_DATA,
            32768, /* PCLKB clock = sub-clock when sub-clock is source clock */
            32768,
            PDL_CGC_SUB_32768
        );

        /* Wait for the Subclock stabilisation time (2 seconds minimum) */
        /* NOTE: As curently running from the Sub-clock the R_CMT_CreateOneShot
        max time limit is > 2 Secs. */
        R_CMT_CreateOneShot(
            0,
            PDL_NO_DATA,
            2.0,
            PDL_NO_FUNC,
            0
        );

        /* Select the HOCO as the clock source */
        R_CGC_Control(
            PDL_CGC_CLK_HOCO,
            PDL_NO_DATA,
            PDL_CGC_RTC_TO_BE_USED
        );
    }

    void Alarm_handler(void)
    {
        uint8_t flags;
        uint32_t time;
        uint32_t date;
        uint8_t buffer[50];

        /* Read time */
        R_RTC_Read(PDL_RTC_READ_CURRENT, &flags, &time, &date);


        sprintf((char*)buffer, "Time = %d%d:%d%d:%d%d \r\n",
                (int)(time & 0xF00000) >> 20,
                (int)(time & 0x0F0000) >> 16,
                (int)(time & 0x00F000) >> 12,
                (int)(time & 0x000F00) >> 8,
                (int)(time & 0x0000F0) >> 4,
                (int)(time & 0x00000F) >> 0);

        R_SCI_Send(RSK_SCI_CHANNEL, PDL_NO_DATA, buffer, 0, PDL_NO_FUNC);

        /* Configure the clock */
```

```
        R_RTC_Control(
            PDL_NO_DATA,
            PDL_RTC_UPDATE_ALARM_TIME,
            PDL_NO_DATA,
            PDL_NO_DATA,
            (time + 0x10),  /* Alarm in another 10 seconds */
            PDL_NO_DATA,
            PDL_NO_DATA, /* Error Adjust */
            PDL_NO_DATA  /* Periodic */
        );

        /* Enter software standby mode ? */
        if(true != bSoftStdbyEnter)
        {
            bSoftStdbyEnter = true;
        }

        nop();

    }
```

Figure 30: Example of using RTC is used and wake up from software standby mode

5.17. Independent Watchdog Timer

Figure 31 shows an example of Independent Watchdog timer usage.

The watchdog timer is configured with a 25% to 75% window and to generate an NMI Interrupt if it does time out.
Because the watchdog timer is not refreshed it does time out and the NMI_handler function is called.

```
/* Peripheral driver function prototypes */
#include "r_pdl_cgc.h"
#include "r_pdl_io_port.h"
#include "r_pdl_intc.h"
#include "r_pdl_iwdt.h"

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

/* Bit mask for underflow and refesh error flags */
#define BIT_MASK_REFEF_UNDFF 0xC000

static void NMI_handler(void);

void main(void)
{
    /* Initialise the system clocks */
    NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.

    /* Enable the IWDTCLK clock */
    R_CGC_Set(
        PDL_CGC_CLK_IWDTLOCO,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA);

    /* Set output pins for LED */
    R_IO_PORT_Set(PDL_IO_PORT_1_5, PDL_IO_PORT_OUTPUT); /* LED1 */

    /* Turn off LED1 */
    R_IO_PORT_Write(PDL_IO_PORT_1_5, 1);

    /* Enable the NMI interrupt for IWDT */
    R_INTC_CreateExtInterrupt(
        PDL_INTC_NMI,
        PDL_INTC_IWDT_ENABLE,
        NMI_handler,
        7);

    /* Configure WDT with a 75% to 25% window, generate NMI on time-out. */
    R_IWDT_Set(PDL_IWDT_TIMEOUT_8192 |
        PDL_IWDT_CLOCK_OCO_16 |
        PDL_IWDT_WIN_START_75 |
        PDL_IWDT_WIN_END_25 |
        PDL_IWDT_TIMEOUT_NMI);

    /* Wait for time-out */
    while (1);

}

static void NMI_handler(void)
{
    uint16_t status = 0;

    /* Read the status of IWDT */
    if(false == R_IWDT_Read(&status))
```

```
        {
            while (1);
        }

        switch(BIT_MASK_REFEF_UNDFF & status)
        {
            case 0x4000:
                /* Turn on LED1 to represent underflow error */
                R_IO_PORT_Modify(PDL_IO_PORT_1_5, PDL_IO_PORT_XOR, 1);
                break;
            default:
                /* Error */
                while(1);
        }

        /* NMI callback function should not return.
          It should stop operation or reset the system. */
        while (1);
    }
```

Figure 31: Example of Independent Watchdog Timer use

5.18. Serial Communication Interface

5.18.1. SCI Asynchronous Using Polling.

This shows the setting of SCI channel 1 and the transmission and reception of data using polling.

```
/* Peripheral driver function prototypes */
#include "r_pdl_cgc.h"
#include "r_pdl_sci.h"

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

volatile uint8_t rx_buffer[5];

void main(void)
{
    /* Initialise the system clocks */
    NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.Please
refer to 5.1 Clock Generation Circuit.

    /* Set pin options */
    R_SCI_Set(1,
        PDL_SCI_PIN_SCI1_RXD1_P30 |
        PDL_SCI_PIN_SCI1_TXD1_P26
    );
    /* Set up SCI channel 1: Async, 8N1, 38400 baud */
    R_SCI_Create(
        1,
        PDL_SCI_ASYNC | PDL_SCI_8N1,
        38400,
        1
    );

    /* Wait while send message */
    R_SCI_Send(
        1,
        PDL_NO_DATA,
        "\r\nHello. Type 5 characters and I will echo them back.\r\n",
        0,
        PDL_NO_FUNC
    );

    /* Wait for message to be sent */
    while(false == data_sent);
    /* Wait for 5 characters to be read */
    R_SCI_Receive(
        1,
        PDL_NO_DATA,
        rx_buffer,
        5,
        PDL_NO_FUNC,
        PDL_NO_FUNC
    );
    /* Echo the 5 characters back. */
    R_SCI_Send(
        1,
        PDL_NO_DATA,
        rx_buffer,
        5,
        PDL_NO_FUNC
    );
    while(1){;}
}
```

Figure 32: Example of SCI asynchronous operation using polling

5.18.2. SCI Asynchronous Using Interrupts.

This shows the setting of SCI channel 1 and the transmission and reception of data using interrupts.

```
/* Peripheral driver function prototypes */
#include "r_pdl_sci.h"
#include "r_pdl_cgc.h"

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

void SCIrx(void);
void SCItx(void);

volatile bool data_received;
volatile bool data_sent;
volatile uint8_t rx_buffer[5];

void main(void)
{
    /* Initialise flags */
    data_sent = false;
    data_received = false;

    /* Initialise the system clocks */
    NOTE: The code to initialise the system clock using R_CGC_Set is omitted here. Please
refer to 5.1 Clock Generation Circuit.

    /* Set channel 1 pin options */
    R_SCI_Set(1,
        PDL_SCI_PIN_SCI1_RXD1_P30 | PDL_SCI_PIN_SCI1_TXD1_P26
    );

    /* Set up SCI channel 1: Async, 8N1, 38400 baud */
    R_SCI_Create(
        1,
        PDL_SCI_ASYNC | PDL_SCI_8N1,
        38400,
        1
    );

    /* Send message - register callback to say when sent */
    R_SCI_Send(
        1,
        PDL_NO_DATA,
        "\r\nHello. Type 5 characters and I will echo them back.\r\n",
        0,
        SCItx
    );

    /* Wait for message to be sent */
    while(false == data_sent);

    /* Start a pending read of 5 characters */
    R_SCI_Receive(
        1,
        PDL_NO_DATA,
        rx_buffer,
        5,
        SCIrx,
        PDL_NO_FUNC
    );

    /* Wait for characters to be received */
    while(false == data_received);
```

```
        /* Echo the 5 characters back. */
        R_SCI_Send(
                    1,
            PDL_NO_DATA,
            rx_buffer,
            5,
            PDL_NO_FUNC
        );

        while(1){;}
    }

    /* Callback function for Rx */
    void SCIrx(void)
    {
        data_received = true;
    }

    /* Callback function for Tx */
    void SCItx(void)
    {
        data_sent = true;
    }
```

Figure 33: Example of SCI Asynchronous operation using interrupts

5.18.3. SCI Asynchronous Using DMAC.

This shows the setting of SCI channel 1 and transmission of data using the DMAC.

```
    /* Peripheral driver function prototypes */
    #include "r_pdl_sci.h"
    #include "r_pdl_cgc.h"
    #include "r_pdl_dmac.h"
    /* RPDL device-specific definitions */
    #include "r_pdl_definitions.h"

    #include <stddef.h>
    #include <string.h>

    const uint8_t* string = "Hello from Renesas RX220 SCI DMAC\r\n";
    uint8_t SCI_status;

    void main(void)
    {
        /* Initialise the system clocks */
        NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.   Please
    refer to 5.1 Clock Generation Circuit.

        /* Set Channel 1 pin options */
        R_SCI_Set(1,
            PDL_SCI_PIN_SCI1_RXD1_P30 |
            PDL_SCI_PIN_SCI1_TXD1_P26
        );

        /* Set up SCI1 : Async, 8N1, 19200 baud */
        R_SCI_Create(
            1,
            PDL_SCI_ASYNC | PDL_SCI_8N1,
            19200,
            1
        );

        /* Configure channel 3 of DMAC to be triggered by SCI1 Tx */
        R_DMAC_Create(
            3,
            PDL_DMAC_REPEAT | PDL_DMAC_SOURCE_ADDRESS_PLUS |
            PDL_DMAC_DESTINATION_ADDRESS_FIXED | PDL_DMAC_SIZE_8,
            PDL_DMAC_TRIGGER_SCI1_TX,
            string, /* Source */
            (const char *)&SCI1.TDR, /* Destination */
            1,
            (uint16_t)strlen((char *)string),
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_FUNC,
            0
        );

        /* Enable DMAC */
        R_DMAC_Control
        (
            3,
            PDL_DMAC_ENABLE,
            PDL_NO_PTR,
            PDL_NO_PTR,
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_DATA
        );
```

```
        /* Start transmission */
        R_SCI_Send(
            1,
            PDL_SCI_DMAC_TRIGGER_ENABLE,
            PDL_NO_PTR,
            PDL_NO_DATA, /* No data as using DMAC */
            PDL_NO_FUNC
        );

        /***********************************************************************
        IMPORTANT: The SCI module does not know when the DMAC has finished,
        therefore, we must tell it using the R_SCI_Control function.
        ***********************************************************************/

        /* Wait for the SCI transmission to end */
        do
        {
            R_SCI_GetStatus
            (
                1,
                &SCI_status,
                PDL_NO_PTR,
                PDL_NO_PTR,
                PDL_NO_PTR
            );
        /* While the 'Transmit status' (BIT_2) is not reporting idle. */
        }while ((SCI_status & 0x04) == 0);

        /* Stop the SCI */
        R_SCI_Control(
            1,
            PDL_SCI_STOP_TX
        );

        /* Send using polling mode */
        R_SCI_Send(
            1,
            PDL_NO_DATA,
            "Hello from Renesas RX220 SCI Polling.\r\n",
            PDL_NO_DATA,
            PDL_NO_FUNC
        );

        while(1){;}
    }
```

Figure 34: Example of SCI Asynchronous operation using DMAC

5.18.4. Synchronous Transmission and Reception

This shows the configuration of SCI channel 6 as the clock master and channel 9 as the slave.
The master transmits data to the slave.
The slave receive function call uses interrupts to call a callback function on completion.

```
/* Peripheral driver function prototypes */
#include "r_pdl_sci.h"
#include "r_pdl_cgc.h"
#include "r_pdl_intc.h"

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

/* SCI channel selection */
#define MASTER_CHANNEL 6
#define SLAVE_CHANNEL  9

/* Rx complete flag */
volatile uint8_t data_received;

/* Callback function prototype */
static void SCI9RxFunc(void);

void main(void)
{
    volatile uint8_t rx_buffer[5];

    /* Initialise the system clocks */
    NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.   Please
refer to 5.1 Clock Generation Circuit.

    /* Set Master (Channel 6) pin options */
    R_SCI_Set(
        MASTER_CHANNEL,
        PDL_SCI_PIN_SCI6_RXD6_PB0 |
        PDL_SCI_PIN_SCI6_TXD6_PB1 |
        PDL_SCI_PIN_SCI6_SCK6_PB3
    );

    /* Set Master (Channel 9) pin options */
    R_SCI_Set(
        SLAVE_CHANNEL,
        PDL_SCI_PIN_SCI9_RXD9_PB6 |
        PDL_SCI_PIN_SCI9_TXD9_PB7 |
        PDL_SCI_PIN_SCI9_SCK9_PB5
    );

    /* Create Master Channel */
    R_SCI_Create(
        MASTER_CHANNEL,
        PDL_SCI_SYNC | PDL_SCI_RX_DISCONNECTED |
        PDL_SCI_CLK_INT_OUT,
        19200,
        1
    );

    /* Create Channel slave */
    /* NOTE: Even though using an external clock the driver needs to know
    the expected baud rate (Bit 31 is set to signify not generating baud) */
    R_SCI_Create(
        SLAVE_CHANNEL,
        PDL_SCI_SYNC | PDL_SCI_TX_DISCONNECTED |
        PDL_SCI_CLK_EXT,
        0x80000000 | 19200,
        1
    );
```

```
        /* Set flag to wait on */
        data_received = false;

        /* Setup a read on channel slave*/
        R_SCI_Receive(
            SLAVE_CHANNEL,
            PDL_NO_DATA,
            rx_buffer,
            5,
            SCI9RxFunc,
            PDL_NO_FUNC
        );

        /* Send the data from the master */
        R_SCI_Send(
            MASTER_CHANNEL,
            PDL_NO_DATA,
            "12345",
            5,
            PDL_NO_FUNC
        );

        /* Wait for channel slave to receive */
        while(data_received == false);

        /* Process the received data here */

        while(1){;}
    }

    /* SCI channel 9 receive complete handler */
    static void SCI9RxFunc(void)
    {
        /* Set flag */
        data_received = true;
    }
```

Figure 35: Example of Synchronous Transmission and Reception code

5.18.5. Synchronous Full Duplex Operation

This shows the configuration of SCI channel 6 as a clock master with both Rx and Tx data pins enabled. Data is received at the same time as data is transmitted.

```
    /* Peripheral driver function prototypes */
    #include "r_pdl_sci.h"
    #include "r_pdl_cgc.h"
    #include "r_pdl_intc.h"

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

    /* SCI channel selection */
    #define MASTER_CHANNEL 6
    #define SLAVE_CHANNEL  9

    #define DATA_LENGTH 5

    /* Complete flags */
    volatile uint8_t data_received;
    volatile uint8_t data_sent;

    /* Callback function prototype */
    static void SCI_Rx_Callback(void);
    static void SCI_Tx_Callback(void);

    void main(void)
    {
        volatile uint8_t rx_buffer[5];

        /* Initialise the system clocks */
        NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.   Please
    refer to 5.1 Clock Generation Circuit.

        /* Set Master (Channel 6) pin options */
        R_SCI_Set(
            MASTER_CHANNEL,
            PDL_SCI_PIN_SCI6_RXD6_PB0 |
            PDL_SCI_PIN_SCI6_TXD6_PB1 |
            PDL_SCI_PIN_SCI6_SCK6_PB3
        );

        /* Set Master (Channel 9) pin options */
        R_SCI_Set(
            SLAVE_CHANNEL,
            PDL_SCI_PIN_SCI9_RXD9_PB6 |
            PDL_SCI_PIN_SCI9_TXD9_PB7 |
            PDL_SCI_PIN_SCI9_SCK9_PB5
        );

        /* Create Clock master channel for Rx and Tx */
        R_SCI_Create(
            MASTER_CHANNEL,
            PDL_SCI_SYNC | PDL_SCI_CLK_INT_OUT |
            PDL_SCI_TX_CONNECTED | PDL_SCI_RX_CONNECTED,
            19200,
            1
        );

        /* Create Slave Channel */
        /* NOTE: Even though using an external clock the driver needs to know
        the expected baud rate (Bit 31 is set to signify not generating baud). */
        R_SCI_Create(
            SLAVE_CHANNEL,
            PDL_SCI_SYNC | PDL_SCI_CLK_EXT,
            0x80000000 | 19200,
```

```
            1
        );

        /* First setup the slave to send. */
        data_sent = false;
        R_SCI_Send(
            SLAVE_CHANNEL,
            PDL_NO_DATA,
            "Slave",
            DATA_LENGTH,
            SCI_Tx_Callback
        );

        /* Setup master to receive. (Non polling) */
        /* NOTE: No clocks pulses will be generated until R_SCI_Send is called. */
        data_received = false;
        R_SCI_Receive(
            MASTER_CHANNEL,
            PDL_NO_DATA,
            rx_buffer,
            DATA_LENGTH,
            SCI_Rx_Callback,
            PDL_NO_FUNC
        );

        /* Dummy send so the Slave Tx and Master Rx will happen. */
        R_SCI_Send(
            MASTER_CHANNEL,
            PDL_NO_DATA,
            "Dummy",
            DATA_LENGTH,
            PDL_NO_FUNC
        );
        /* Wait for Rx to finish */
        while(data_received == false);

        /* Process the received data here */
        while(1){;}
    }

    /* Callback function for Rx */
    static void SCI_Rx_Callback(void)
    {
        data_received = true;
    }

    /* Callback function for Tx */
    static void SCI_Tx_Callback(void)
    {
        data_sent = true;
    }
```

Figure 36: Example of Synchronous Full Duplex operation

5.18.6. SCI Reception in Asynchronous Multi-Processor mode

This shows the setting of SCI channel 9 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_intc.h"

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

    void SCIrx(void);
    void SCIEr(void);

    #define NUM_DATA 50
    volatile uint8_t data_received;
    volatile uint8_t error_happen;
    volatile uint8_t receive_data_0[NUM_DATA];
    volatile uint8_t receive_data[NUM_DATA];

    void main(void)
    {
        uint8_t i;
        bool id_received;

        /* Initialise the system clocks */
        NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.   Please
    refer to 5.1 Clock Generation Circuit.

        /* Initialise the receive buffer. */
        for (i=0; i<NUM_DATA; i++)
        {
            receive_data[i] = 0;
        }

        /* Set Channel 9  pin options */
        R_SCI_Set(
            9,
            PDL_SCI_PIN_SCI9_RXD9_PB6 |
            PDL_SCI_PIN_SCI9_TXD9_PB7
        );

        /* Configure the RS232 port, specify Async MP mode */
        R_SCI_Create(
            9,
            PDL_SCI_8N1 | PDL_SCI_ASYNC_MP,
            9600,
            15
        );

        /* ------------------------------------------- */
        /*   Async MP mode, data Reception, by CPU ISR  */
        /* ------------------------------------------- */

        data_received = false;
        error_happen = false;

        /* Wait by CPU ISR, until receive matching Station ID (0x0A) */
        R_SCI_Receive(
            9,
            0x0A00 | PDL_SCI_MP_ID_CYCLE,
            PDL_NO_PTR,
            0,
            SCIrx,
            SCIEr
```

```
        );

        while (data_received == false);

        data_received = false;

        /* Receive data (ID = 0x0A) by CPU ISR */
        R_SCI_Receive(
            9,
            PDL_NO_DATA,
            receive_data,
            29,
            SCIrx,
            SCIEr
        );

        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(
            9,
            0x0100 | PDL_SCI_MP_ID_CYCLE,
            PDL_NO_PTR,
            0,
            PDL_NO_FUNC,
            SCIEr
        );

        if (id_received == true)
        {
            /* Receive data (ID = 0x01) by polling */
            R_SCI_Receive(
                9,
                PDL_NO_DATA,
                receive_data,
                21,
                PDL_NO_FUNC,
                SCIEr
            );
        }
    }

    void SCIrx(void)
    {
        data_received = true;
    }

    void SCIEr(void)
    {
        error_happen = true;
    }
```

Figure 37: Example of SCI Reception code in Asynchronous Multi-Processor mode

5.18.7. SCI Transmission in Asynchronous Multi-Processor mode

This shows the setting of SCI channel 9 and the Multi-Processor mode transmission of data using interrupts and polling.

```
    /* PDL functions */
    #include "r_pdl_sci.h"
    #include "r_pdl_cgc.h"
    #include "r_pdl_intc.h"

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

    void SCItx(void);

    uint8_t* send_data0 = "Welcome to the Renesas RX220.\n\r";
    uint8_t* send_data = "testing ASYNC MP mode";
    bool tx_end;

    volatile bool g_Switch1Pressed;
    static void SW1_handler(void);

    void main(void)
    {
        /* Initialise the system clocks */
        NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.   Please
    refer to 5.1 Clock Generation Circuit.

        /* Set Channel 9  pin options */
        R_SCI_Set(
            9,
            PDL_SCI_PIN_SCI9_RXD9_PB6 |
            PDL_SCI_PIN_SCI9_TXD9_PB7
        );

        /* Configure the RS232 port, specify Async MP mode */
        R_SCI_Create(
            9,
            PDL_SCI_8N1 | PDL_SCI_ASYNC_MP,
            9600,
            15
        );

        /* ------------------------------------------- */
        /* Async MP mode, data Transmission, by CPU ISR */
        /* ------------------------------------------- */

        NOTE: The receiving side must be ready before this ID is transmitted.

        /* Send Target Station ID (0x0A), by internal polling */
        R_SCI_Send(
            9,
            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(
            9,
            PDL_NO_DATA,
            send_data0,
            0,
            SCItx
```

```
        );

        while(tx_end == false);

        /* ----------------------------------------- */
        /* Async MP mode, data Transmission, by polling */
        /* ----------------------------------------- */

        NOTE: The receiving side must be ready before this ID is transmitted.

        /* Send Target Station ID (0x01) by internal polling */
        R_SCI_Send(
            9,
            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(
            9,
            PDL_NO_DATA,
            send_data,
            0,
            PDL_NO_FUNC
        );

        while(1){;}
    }

    void SCItx(void)
    {
        tx_end = true;
    }
```

Figure 38: Example of SCI Transmission code in Asynchronous Multi-Processor mode

5.18.8. SCI in SPI Mode

This shows the setting of SCI channel 6 in to SPI master mode and the transmission of data using interrupts.

```
/* PDL functions */
#include "r_pdl_sci.h"
#include "r_pdl_cgc.h"

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

static void SCItx(void);

volatile bool data_sent = false;

void main(void)
{
    /* Initialise the system clocks */
    NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.   Please
refer to 5.1 Clock Generation Circuit.

    /* Set Channel 6  pin options */
    R_SCI_Set(
        6,
        PDL_SCI_PIN_SCI6_RXD6_PB0 |
        PDL_SCI_PIN_SCI6_TXD6_PB1 |
        PDL_SCI_PIN_SCI6_SCK6_PB3
    );

    /* Create SPI master */
    R_SCI_Create(
        6,
        PDL_SCI_SYNC | PDL_SCI_SPI_MODE |
        PDL_SCI_RX_DISCONNECTED | PDL_SCI_CLK_INT_OUT,
        19200,
        1
    );

    /* Start sending data */
    R_SCI_SPI_Transfer(
        6,
        PDL_NO_DATA,
        5,
         "12345",
        SCItx,
        PDL_NO_DATA,
        PDL_NO_FUNC,
        PDL_NO_FUNC
    );

    /* Wait for data to be sent */
    while(data_sent == false);

    /* Close this channel */
    R_SCI_Destroy(6);
}

static void SCItx(void)
{
    data_sent = true;
}
```

Figure 39: Example of SCI in SPI mode

5.18.9. SCI in IIC Mode

This shows the setting of SCI channel 9 into IIC mode and then a write and read to an IIC EEPROM.

```
    /* PDL functions */
    #include "r_pdl_sci.h"
    #include "r_pdl_cgc.h"
    #include "r_pdl_cmt.h"

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

    /* SCI IIC Channel */
    #define CHANNEL_SCI_IIC 9
    /* IIC Slave address of EEPROM */
    #define SLAVE_ADDRESS 0xA0
    /* Address in EEPROM where we will write a byte */
    #define EEPROM_ADDRESS 0x01
    /* Value to be written to the EEPROM */
    #define EEPROM_VALUE 0xAA

    void main(void)
    {
        /* Data Buffer */
        volatile uint8_t IIC_Buffer[10];

        /* Initialise the system clocks */
        NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.   Please
    refer to 5.1 Clock Generation Circuit.

        /* Set Channel 9  pin options */
        R_SCI_Set(
            CHANNEL_SCI_IIC,
            PDL_SCI_PIN_SCI9_RXD9_PB6 |
            PDL_SCI_PIN_SCI9_TXD9_PB7
        );

        /* Configure the SCI IIC Channel */
        R_SCI_Create(
            CHANNEL_SCI_IIC,
            PDL_SCI_SYNC |
            PDL_SCI_IIC_MODE |
            PDL_SCI_IIC_DELAY_SDA_20_21,
            9600,
            1
        );

        /* Set up data buffer for the write. */
        /* Address in EEPROM */
        IIC_Buffer[0] = EEPROM_ADDRESS;
        /* Data to write */
        IIC_Buffer[1] = EEPROM_VALUE;

        /* IIC write */
        R_SCI_IIC_Write(
            CHANNEL_SCI_IIC,
            PDL_NO_DATA,
            SLAVE_ADDRESS,
            2,
            IIC_Buffer,
            PDL_NO_FUNC
        );

        /* Wait for 5ms while the EEPROM updates */
        R_CMT_CreateOneShot(
            0,
            0,
```

```
            5E-3,
            PDL_NO_FUNC,
            0
        );

        /* Confirm this write worked by reading back the data from the EEPROM. */
        /* 1. Set current EEPROM address*/
        IIC_Buffer[0] = EEPROM_ADDRESS;
        R_SCI_IIC_Write(
            CHANNEL_SCI_IIC,
            PDL_NO_DATA,
            SLAVE_ADDRESS,
            1,
            IIC_Buffer,
            PDL_NO_FUNC
        );

        /* 2. Read data from current address */
        R_SCI_IIC_Read(
            CHANNEL_SCI_IIC,
            PDL_NO_DATA,
            SLAVE_ADDRESS,
            1,
            IIC_Buffer,
            PDL_NO_FUNC
        );

        /* Confirm the value written is the same as the value read */
        if(IIC_Buffer[0] != EEPROM_VALUE)
        {
            /* User Handle Error */
        }
    }
```

Figure 40: Example of SCI in IIC mode

5.18.10. SCI in IIC Mode using DMAC

This shows the setting of SCI channel 9 in to IIC mode and then a write to an IIC EEPROM using the DMAC.

```
    /* PDL functions */
    #include "r_pdl_sci.h"
    #include "r_pdl_cgc.h"
    #include "r_pdl_dmac.h"

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

    static void Callback(void);

    /* SCI IIC Channel */
    #define CHANNEL_SCI_IIC 9
    /* IIC Slave address of EEPROM */
    #define SLAVE_ADDRESS 0xA0
    /* Address in EEPROM where we will write a byte */
    #define EEPROM_ADDRESS 0x01

    volatile bool data_sent = false;

    void main(void)
    {
        /* Data Buffer */
        volatile uint8_t IIC_Buffer[10];

        /* Initialise the system clocks */
        NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.   Please
    refer to 5.1 Clock Generation Circuit.

        /* Set Channel 9  pin options */
        R_SCI_Set(
            CHANNEL_SCI_IIC,
            PDL_SCI_PIN_SCI9_RXD9_PB6 |
            PDL_SCI_PIN_SCI9_TXD9_PB7
        );

        /* Configure the SCI IIC Channel */
        R_SCI_Create(
            CHANNEL_SCI_IIC,
            PDL_SCI_SYNC |
            PDL_SCI_IIC_MODE |
            PDL_SCI_IIC_DELAY_SDA_20_21,
            9600,
            1
        );

        /* Setup date to write to EEPROM */
        /* Address in EEPROM */
        IIC_Buffer[0] = EEPROM_ADDRESS;
        /* Data to store in EEPROM */
        IIC_Buffer[1] = 1;
        IIC_Buffer[2] = 2;
        IIC_Buffer[3] = 3;
        IIC_Buffer[4] = 4;
        IIC_Buffer[5] = 5;

        /* Setup DMAC to write data to IIC */
        /* Configure channel 3 of DMAC to be triggered by SCI9 Tx */
        R_DMAC_Create(
            3,
            PDL_DMAC_REPEAT | PDL_DMAC_SOURCE_ADDRESS_PLUS | \
            PDL_DMAC_DESTINATION_ADDRESS_FIXED |PDL_DMAC_SIZE_8 |PDL_DMAC_IRQ_END,
            PDL_DMAC_TRIGGER_SCI9_TX,
            IIC_Buffer,              /* Source */
```

```
            (uint8_t *)&SCI9.TDR,     /* Dest */
            1,
            6,                        /* Data length (Address in EEPROM + 5 Data) */
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_DATA,
            Callback,                 /* Callback done function */
            7                         /* Interrupt priority */
        );

        /* Enable DMAC channel 3 */
        R_DMAC_Control(
            3,
            PDL_DMAC_ENABLE,
            PDL_NO_PTR,
            PDL_NO_PTR,
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_DATA
        );

        /* Clear flag */
        data_sent = false;
        /* Start IIC Write */
        R_SCI_IIC_Write(
            CHANNEL_SCI_IIC,
            PDL_SCI_IIC_DMAC_TRIGGER_ENABLE,
            SLAVE_ADDRESS,
            PDL_NO_DATA, /* No data length as using DMAC */
            PDL_NO_DATA, /* No buffer as using DMAC */
            PDL_NO_FUNC
        );

        /* Wait for write to complete */
        while(false == data_sent){;}

        /* Because using DMAC need to manually send a stop to end the transfer */
        R_SCI_Control(
            CHANNEL_SCI_IIC,
            PDL_SCI_IIC_STOP
        );
    }

    /* Callback done */
    static void Callback(void)
    {
        data_sent = true;
    }
```

Figure 41: Example of SCI in IIC mode using DMAC

5.18.11. SCI in IIC Mode using DTC

This shows the setting of SCI channel 9 in to IIC mode and then a read from an IIC EEPROM using the DTC.

```
/* PDL functions */
#include "r_pdl_sci.h"
#include "r_pdl_cgc.h"
#include "r_pdl_dtc.h"

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

static void CallbackRx(void);

/* SCI IIC Channel */
#define CHANNEL_SCI_IIC 9
/* IIC Slave address of EEPROM */
#define SLAVE_ADDRESS 0xA0
/* Address in EEPROM where we will write a byte */
#define EEPROM_ADDRESS 0x01

/* Flag */
volatile uint8_t data_received;

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

void main(void)
{
    /* Data Buffer */
    volatile uint8_t IIC_Buffer[10];

    /* DTC needs to write dummy data to SCI.TDR when reading. */
    uint8_t IIC_Dummy_value = 0xFF;

    /* 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];

    /* Initialise the system clocks */
    NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.   Please
refer to 5.1 Clock Generation Circuit.

    /* Set Channel 9  pin options */
    R_SCI_Set(
        CHANNEL_SCI_IIC,
        PDL_SCI_PIN_SCI9_RXD9_PB6 |
        PDL_SCI_PIN_SCI9_TXD9_PB7
    );

    /* Setup the SCI IIC channel */
    R_SCI_Create(
        CHANNEL_SCI_IIC,
        PDL_SCI_SYNC | PDL_SCI_IIC_MODE | PDL_SCI_IIC_DELAY_SDA_20_21,
        9600,
        1
    );

    /* Configure the DTC controller */
    R_DTC_Set(
        PDL_DTC_ADDRESS_FULL,
        dtc_vector_table
    );

    /* Set current EEPROM address */
    IIC_Buffer[0] = EEPROM_ADDRESS;
```

```
        /* Use blocking function for this, DTC will be used for the data part. */
        R_SCI_IIC_Write(
            CHANNEL_SCI_IIC,
            PDL_SCI_IIC_NOSTOP,
            SLAVE_ADDRESS,
            1,
            IIC_Buffer,
            PDL_NO_FUNC
        );

        /* Set flag */
        data_received = false;

        /* Read data from current EEPROM address using DTC */
        /* Start with an IIC Re-start */
        /* DTC on Rx */
        R_DTC_Create(
            PDL_DTC_NORMAL | PDL_DTC_DESTINATION_ADDRESS_PLUS | \
            PDL_DTC_SOURCE_ADDRESS_FIXED |PDL_DTC_SIZE_8 |PDL_DTC_IRQ_COMPLETE |
            PDL_DTC_TRIGGER_RXI9,
            dtc_iic1_rx_transfer_data,
            (uint8_t *)&SCI9.RDR,     /* Source */
            IIC_Buffer,               /* Destination */
            /* Data length is one less than we want to read as
            use R_SCI_IIC_ReadLastByte */
            4,
            PDL_NO_DATA
        );

        /* DTC on Tx (To write the dummy data out.) */
        /* Data length is 2 less than we want to read as first dummy byte
        is written out by R_SCI_IIC_Read function and last one when we use
        R_SCI_IIC_ReadLastByte. */
        R_DTC_Create(
            PDL_DTC_NORMAL | PDL_DTC_SOURCE_ADDRESS_FIXED |
            PDL_DTC_DESTINATION_ADDRESS_FIXED |PDL_DTC_SIZE_8 | \
            PDL_DTC_IRQ_COMPLETE |PDL_DTC_TRIGGER_TXI9,
            dtc_iic1_tx_transfer_data,
            &IIC_Dummy_value,         /* Source */
            (uint8_t *)&SCI9.TDR,     /* Destination */
            3,             /* Data length */
            PDL_NO_DATA
        );

        /* Enable the DTC */
        R_DTC_Control(
            PDL_DTC_START,
            PDL_NO_PTR,
            PDL_NO_PTR,
            PDL_NO_PTR,
            PDL_NO_DATA,
            PDL_NO_DATA
        );

        /* Start the IIC Read */
        R_SCI_IIC_Read(
            CHANNEL_SCI_IIC,
            PDL_SCI_IIC_RESTART | PDL_SCI_IIC_DTC_TRIGGER_ENABLE,
            SLAVE_ADDRESS,
            PDL_NO_DATA, /* No data length as using DTC */
            PDL_NO_DATA, /* No buffer as using DTC */
            CallbackRx
        );

        /* Wait for rx */
        while(data_received == false){;}

        /* Because using DMAC need to manually get the last byte.
```

```
        This will also generate the stop condition. */
        R_SCI_IIC_ReadLastByte(
            CHANNEL_SCI_IIC,
            &IIC_Buffer[4]
        );
    }

    /* Callback function for Rx */
    static void CallbackRx(void)
    {
        data_received = true;
    }
```

Figure 42: Example of SCI in IIC mode using DTC

5.19. I²C Bus Interface

In the following examples, the bus activity will be illustrated using the following format.

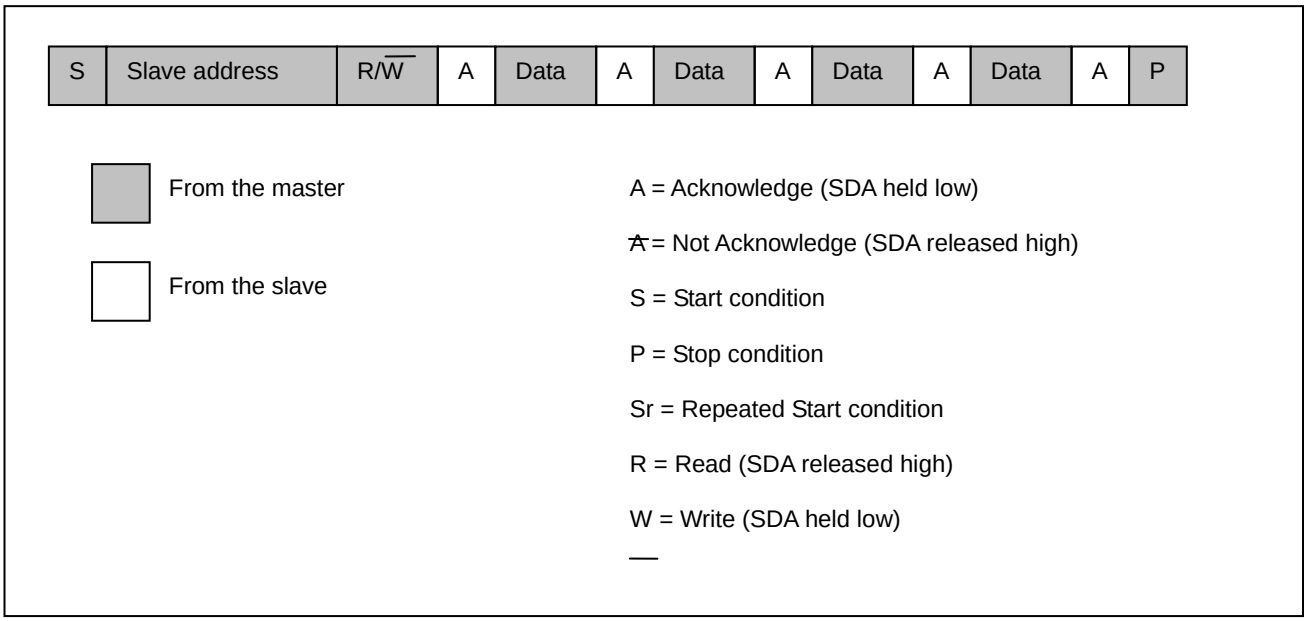


Figure 43: I²C bus activity notation

5.19.1. 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 I²C channel 0 will be configured for Master operation and used to send 4 bytes to a slave.

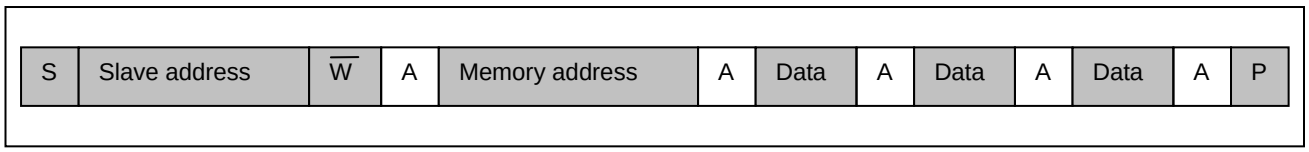


Figure 44: The bus activity, showing 4 bytes being transmitted to the EEPROM

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

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

#define EEPROM_ADDRESS 0xA0

void main(void)
{
    const uint8_t eeprom_data_array_1[4] = {0x00, 0x01, 0x02, 0x03};
    uint8_t data_storage[3];
    uint32_t status_flags = 0;
    uint16_t TxChars;

    /* Initialise the system clocks */
    NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.

    /* Select IIC Pins */
    R_IIC_Set(
        PDL_IIC_PIN_SDA_P13 | PDL_IIC_PIN_SCL_P12);


    /* 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 sub 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 45: Configure the I²C channel and write 3 data bytes to the first locations

2) Reception

Continuing from above, the memory address pointer of an EEPROM will be modified, and then a Repeat Start condition used to change to read from that memory location in the EEPROM.

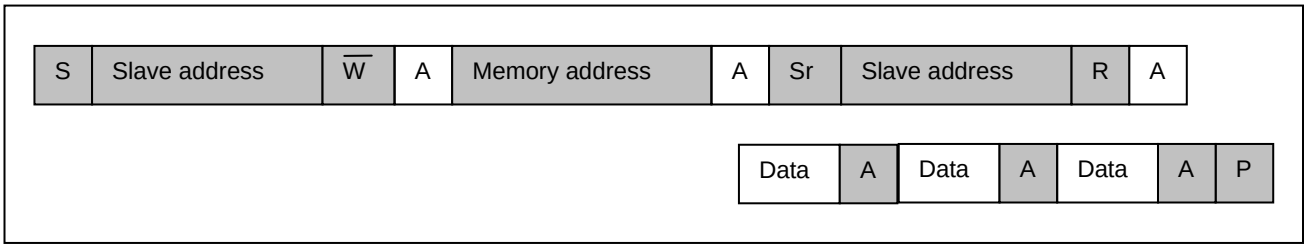


Figure 46: The bus activity, showing the Repeated Start condition when switching to the Read process

```
        /* Send 1 byte to the EEPROM to update the EEPROM sub address bits
             and do not stop */
        R_IIC_MasterSend(
            0,
            PDL_IIC_STOP_DISABLE,
            EEPROM_ADDRESS,
            eeprom_data_array_1,
            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,
            3,
            PDL_NO_FUNC,
            0
        );
```

Figure 47: Set the EEPROM sub address and then read 2 bytes

5.19.2. Master mode with DMAC

In the following example, data is written to an EEPROM in two bursts. DMAC channel 3 is used to handle the data transfer. The same EEPROM address locations are then read out in two bursts. DMAC channel 2 is used to handle the data transfer.

```
/* PDL functions */
#include "r_pdl_cgc.h"
#include "r_pdl_iic.h"
#include "r_pdl_cmt.h"
#include "r_pdl_dmac.h"

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

static void write_eeprom_data(void);
static void read_eeprom_data(void);
void iic_tx_dmac_end_handler(void);
void iic_rx_dmac_end_handler(void);

#define EEPROM_MEMORY_ADDRESS_UPPER 0x00
#define EEPROM_MEMORY_ADDRESS_LOWER 0x00
#define EEPROM_ADDRESS (0x00A0 | EEPROM_MEMORY_ADDRESS_UPPER)

#define IIC_CHANNEL 0

volatile uint8_t bus_busy;
volatile uint8_t data_storage[20];

void main(void)
{
    #define ARRAY_1_SIZE 6 /* 5 Data bytes + 1 address */
    #define ARRAY_2_SIZE 11 /* 10 Data bytes + 1 address */
    const uint8_t eeprom_data_array_1[ARRAY_1_SIZE] = {EEPROM_MEMORY_ADDRESS_LOWER,
0x11, 0x22, 0x33, 0x44, 0x55};
    const uint8_t eeprom_data_array_2[ARRAY_2_SIZE] = {EEPROM_MEMORY_ADDRESS_LOWER + 5,
0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F};
    uint8_t i;

    /* Initialise the system clocks */
    NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.
    /* Select IIC Pins */
    R_IIC_Set(
        PDL_IIC_PIN_SDA_P13 | PDL_IIC_PIN_SCL_P12);

    /* Set up a DMAC channel for IIC transmission */
        R_DMAC_Create(
        3,
        PDL_DMAC_NORMAL | PDL_DMAC_SIZE_8 |
            PDL_DMAC_SOURCE_ADDRESS_PLUS |
            PDL_DMAC_DESTINATION_ADDRESS_FIXED |
            PDL_DMAC_IRQ_END,
        PDL_DMAC_TRIGGER_IIC0_TX,
        eeprom_data_array_1,
        (uint8_t *)&RIIC0.ICDRT,
        ARRAY_1_SIZE,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        iic_tx_dmac_end_handler,
        7
    );

    /* Set up a DMAC channel for IIC reception*/
    /* This will read back the bytes previously written except the last one
    which will be read using R_IIC_MasterReceiveLast */
        R_DMAC_Create(
```

```
        2,
        PDL_DMAC_NORMAL | PDL_DMAC_SIZE_8 |
            PDL_DMAC_SOURCE_ADDRESS_FIXED |
            PDL_DMAC_DESTINATION_ADDRESS_PLUS |
            PDL_DMAC_IRQ_END,
        PDL_DMAC_TRIGGER_IIC0_RX,
        (uint8_t *)&RIIC0.ICDRR,
        data_storage,
        ARRAY_1_SIZE-2, /* Array size written - sub address byte - last byte */
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        iic_rx_dmac_end_handler,
        7
    );

    /* Select I²C mode at 100kHz, 300ns rise time, 200ns fall time */
    R_IIC_Create(
        IIC_CHANNEL,
        PDL_IIC_MODE_IIC | PDL_IIC_INT_PCLK_DIV_8,
        0,
        0,
        0,
        0,
        100E3,
        (300 << 16) | 200
    );

    /* Write the data into the EEPROM */
    write_eeprom_data();

        /* Prepare the next data for writing to the EEPROM */
        R_DMAC_Control(
        3,
        PDL_DMAC_SUSPEND | PDL_DMAC_ENABLE | \
        PDL_DMAC_UPDATE_SOURCE | PDL_DMAC_UPDATE_COUNT | PDL_DMAC_CLEAR_DTIF,
        eeprom_data_array_2,
        PDL_NO_PTR,
        ARRAY_2_SIZE,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        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(
        IIC_CHANNEL,
        PDL_IIC_STOP_DISABLE,
        EEPROM_ADDRESS,
        eeprom_data_array_1,
        1,
        PDL_NO_FUNC,
        0
    );

    /* Read data from the EEPROM using the DMAC */
    read_eeprom_data();

    /* Prepare to read the next data */
    /* This will read back the bytes previously written except the last one
```

```
        which will be read using R_IIC_MasterReceiveLast */
            R_DMAC_Control(
            2,
            PDL_DMAC_SUSPEND | PDL_DMAC_ENABLE | \
             PDL_DMAC_UPDATE_DESTINATION | PDL_DMAC_UPDATE_COUNT,
            PDL_NO_PTR,
            &data_storage[ARRAY_1_SIZE-1],
            ARRAY_2_SIZE-2, /* Array size written - sub address byte - last byte */
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_DATA,
            PDL_NO_DATA
        );

        /* Read data from the EEPROM using the DMAC */
        read_eeprom_data();
    }

    static void write_eeprom_data(void)
    {
        bus_busy = true;
        /* Send data to the EEPROM using the DMAC */
        R_IIC_MasterSend(
            IIC_CHANNEL,
            PDL_IIC_DMAC_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_eeprom_data(void)
    {
        bus_busy = true;
        /* Read data from the EEPROM using the DMAC */
        R_IIC_MasterReceive(
            IIC_CHANNEL,
            PDL_IIC_DMAC_TRIGGER_ENABLE,
            EEPROM_ADDRESS,
            PDL_NO_PTR,
            0,
            PDL_NO_FUNC,
            0
        );

        while (bus_busy == true);
    }

    void iic_tx_dmac_end_handler(void)
    {
        uint32_t status_flags = 0;

        /* Wait for the transmission to complete */
        do
        {
```

```
            R_IIC_GetStatus(
                IIC_CHANNEL,
                &status_flags,
                PDL_NO_PTR,
                PDL_NO_PTR
            );
        } while((status_flags & 0x0080u) == 0x0u);

        /* Issue a Stop condition */
        R_IIC_Control(
            IIC_CHANNEL,
            PDL_IIC_STOP
        );

        bus_busy = false;
    }

    void iic_rx_dmac_end_handler(void)
    {
        uint32_t DestAddr = 0;

        /* Read the next destination address for the current transfer */
        R_DMAC_GetStatus(
            2,
            PDL_NO_PTR,
            PDL_NO_PTR,
            &DestAddr,
            PDL_NO_PTR,
            PDL_NO_PTR
        );

        /* Read one more byte with NACK condition and stop */
        R_IIC_MasterReceiveLast(
            IIC_CHANNEL,
            (uint8_t *)DestAddr
        );

        bus_busy = false;
    }
```

Figure 48: An example of writing data to and reading data from an EEPROM, using two DMAC channels

5.19.3. 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"

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

static void write_eeprom_data(void);
static void read_eeprom_data(void);
void iic_tx_end_handler(void);
void iic_rx_end_handler(void);

#define EEPROM_MEMORY_ADDRESS_UPPER 0x00
#define EEPROM_MEMORY_ADDRESS_LOWER 0x00
#define EEPROM_ADDRESS (0x00A0 | EEPROM_MEMORY_ADDRESS_UPPER)

#define IIC_CHANNEL 0

/* The Tx callback must process the following states */
typedef enum IIC_TX_STATE
{
    IIC_TX_STATE_FINISHED,
    IIC_TX_STATE_WAIT_DTC,
    IIC_TX_STATE_WAIT_LAST_BYTE
}IIC_TX_STATE;

static IIC_TX_STATE g_IIC_Tx_State = IIC_TX_STATE_FINISHED;

volatile uint8_t g_IIC_Rx_busy;
volatile uint8_t data_storage[20];

/* Reserve an area for the DTC vector table */
#pragma address dtc_vector_table = 0x00002000
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)
{
    #define ARRAY_1_SIZE 6 /* 5 Data + 1 address */
    #define ARRAY_2_SIZE 11 /* 10 Data + 1 address */
    const uint8_t eeprom_data_array_1[ARRAY_1_SIZE] = {EEPROM_MEMORY_ADDRESS_LOWER,
0x11, 0x22, 0x33, 0x44, 0x55};
    const uint8_t eeprom_data_array_2[ARRAY_2_SIZE] = {EEPROM_MEMORY_ADDRESS_LOWER + 5,
0x66, 0x77, 0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF};
    uint8_t i;

    /* Initialise the system clocks */
    NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.

    /* Select IIC Pins */
    R_IIC_Set(
        PDL_IIC_PIN_SDA_P13 | PDL_IIC_PIN_SCL_P12);


    /* 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_IIC0_TX ,
        dtc_iic1_tx_transfer_data,
        eeprom_data_array_1,
        (uint8_t *)&RIIC0.ICDRT,
        ARRAY_1_SIZE,
        PDL_NO_DATA
    );

    /* Set up a DTC channel for IIC reception */
    /* This will read back the bytes previously written except the last one
    which will be read using R_IIC_MasterReceiveLast */
    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_IIC0_RX,
        dtc_iic1_rx_transfer_data,
        (uint8_t *)&RIIC0.ICDRR,
        data_storage,
        ARRAY_1_SIZE-2, /* Array size written - sub address byte - last byte */
        PDL_NO_DATA
    );

    /* Select I²C mode at 100kHz, 300ns rise time, 200ns fall time */
    R_IIC_Create(
        IIC_CHANNEL,
        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 to write to 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,
        ARRAY_2_SIZE,
        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(
        IIC_CHANNEL,
        PDL_IIC_STOP_DISABLE,
        EEPROM_ADDRESS,
        eeprom_data_array_1,
        1,
        PDL_NO_FUNC,
        0
    );

    /* Read data from the EEPROM using the DTC */
    read_eeprom_data();

    /* Prepare to read the next data */
    R_DTC_Control(
        PDL_DTC_UPDATE_DESTINATION | PDL_DTC_UPDATE_COUNT,
        dtc_iic1_rx_transfer_data,
        PDL_NO_PTR,
        &data_storage[ARRAY_1_SIZE-1],
        ARRAY_2_SIZE-2, /* Array size written - sub address byte - last byte */
        PDL_NO_DATA
    );

    /* Read data from the EEPROM using the DTC */
    read_eeprom_data();
}

static void write_eeprom_data(void)
{
    /* Set state variable so callback function will no how to behave. */
    g_IIC_Tx_State = IIC_TX_STATE_WAIT_DTC;

    /* Send data to the EEPROM using the DTC */
    R_IIC_MasterSend(
        IIC_CHANNEL,
        PDL_IIC_DTC_TRIGGER_ENABLE,
        EEPROM_ADDRESS,
        PDL_NO_PTR,
        0,
        iic_tx_end_handler,
        7
    );

    while (g_IIC_Tx_State != IIC_TX_STATE_FINISHED)
    {
        uint32_t iic_flags;
        uint16_t flags;
        uint32_t src;
        uint32_t dest;
        uint16_t counter;

        R_DTC_GetStatus(dtc_iic1_tx_transfer_data,
                        &flags,
                        &src,
                        &dest,
                        &counter,
                        PDL_NO_PTR);

        R_IIC_GetStatus(
            IIC_CHANNEL,
```

```
                &iic_flags,
                PDL_NO_PTR,
                PDL_NO_PTR
            );
        }

        /* Wait for 5ms while the EEPROM updates */
        R_CMT_CreateOneShot(
            0,
            0,
            5E-3,
            PDL_NO_FUNC,
            0
        );
    }

    static void read_eeprom_data(void)
    {
        g_IIC_Rx_busy = true;

        /* Read data from the EEPROM using the DTC */
        R_IIC_MasterReceive(
            IIC_CHANNEL,
            PDL_IIC_DTC_TRIGGER_ENABLE,
            EEPROM_ADDRESS,
            PDL_NO_PTR,
            0,
            iic_rx_end_handler,
            7
        );
        while (g_IIC_Rx_busy == true);
    }

    /* This callback is registerd with R_IIC_MasterSend. */
    /* We have configured the DTC to pass on the interrupt when it has transfered
    all data to the IIC, so this will be called then.
    This will also get called when the last byte has been transmitted by the IIC.
    When the last byte has been sent it is our job to issue a IIC stop. */
    void iic_tx_end_handler(void)
    {
        /* Process according to state. */
        switch(g_IIC_Tx_State)
        {
            case IIC_TX_STATE_WAIT_DTC:
                /* DTC has finished transfer so now wait for last byte to be transmitted. */
                g_IIC_Tx_State = IIC_TX_STATE_WAIT_LAST_BYTE;
                break;
            case IIC_TX_STATE_WAIT_LAST_BYTE:
            {
                uint32_t status_flags = 0;

                /* Wait for the transmission to fully complete */
                do
                {
                    R_IIC_GetStatus(
                        IIC_CHANNEL,
                        &status_flags,
                        PDL_NO_PTR,
                        PDL_NO_PTR
                    );
                } while((status_flags & 0x0080u) == 0x0u);

                /* Issue a Stop condition */
                R_IIC_Control(
                    IIC_CHANNEL,
                    PDL_IIC_STOP
                );
```

```
                /* This master write has completed */
                g_IIC_Tx_State = IIC_TX_STATE_FINISHED;
                break;
            }
            default:
                ;/*Not expected*/
        };
    }

    /* This callback is registerd with R_IIC_MasterReceive. */
    void iic_rx_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 and stop */
        R_IIC_MasterReceiveLast(
            IIC_CHANNEL,
            (uint8_t *)DestAddr
        );

        g_IIC_Rx_busy = false;
    }
```

Figure 49: An example of writing data to and reading data from an EEPROM, using the DTC

5.19.4. Slave mode

In this example the MCU behaves as a virtual slave memory device on channel 0.
It will respond to 7-bit address 0001001b.
The sample is interrupt driven after the initial setup.

```
/* Peripheral driver function prototypes */
#include "r_pdl_iic.h"
#include "r_pdl_cgc.h"

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

/* Define the size of the virtual memory */
#define STORAGE_SIZE 0x100
#define RX_BUFFER_SIZE (STORAGE_SIZE + 1)

#define SLAVE_CHANNEL 0
#define SLAVE_ADDRESS 0xB0

static void slave_callback(void);
static void StoreData(uint16_t count);

/* Current memory address */
volatile uint8_t data_storage_index = 0;
volatile uint8_t data_storage[STORAGE_SIZE];
volatile uint8_t Rx_Buffer[RX_BUFFER_SIZE];

void main(void)
{
    /* Initialise the system clocks */
    NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.

    /* Select IIC Pins */
    R_IIC_Set(
        PDL_IIC_PIN_SDA_P13 | PDL_IIC_PIN_SCL_P12);

    /* Select IIC mode at 100kHz */
    R_IIC_Create(
        SLAVE_CHANNEL,
        PDL_IIC_MODE_IIC | PDL_IIC_INT_PCLK_DIV_8,
        PDL_IIC_SLAVE_0_ENABLE_7,
        SLAVE_ADDRESS,
        PDL_NO_DATA,
        PDL_NO_DATA,
        100E3,
        0
    );

    /* Start monitor the channel */
    R_IIC_SlaveMonitor(
        SLAVE_CHANNEL,
        PDL_NO_DATA,
        Rx_Buffer,
        RX_BUFFER_SIZE,
        slave_callback,
        7
    );

    /* The rest is interrupt driven */
    while(1);
}

/* R_IIC_SlaveMonitor or R_IIC_SlaveSend callback */
static void slave_callback(void)
{
    uint32_t status_flags = 0;
```

```
        uint16_t tx_count = 0;
        uint16_t rx_count = 0;
        bool bStartMonitor = true;

        /* Read the status */
        R_IIC_GetStatus(
            SLAVE_CHANNEL,
            &status_flags,
            &tx_count,
            &rx_count
        );

        /* Has the master just completed a write? */
        if(rx_count != 0)
        {
            StoreData(rx_count);

            /* Start monitoring again. */
            bStartMonitor = true;
        }
        /* Has the master just completed a read? */
        else if(tx_count != 0)
        {
            /* Increment the current index by the amount the master read */
            data_storage_index += tx_count;

            /* Start monitoring again. */
            bStartMonitor = true;
        }
        /* Is the master starting a read?
           Check this by seeing if in transmit mode. */
        else if(0 != (status_flags & BIT_6))
        {
            /* Send data to master based on current address */
            R_IIC_SlaveSend(
                SLAVE_CHANNEL,
                &data_storage[data_storage_index],
                (uint16_t)(STORAGE_SIZE - data_storage_index)
            );

            /* Don't start monitoring again until the R_IIC_SlaveSend completes. */
            bStartMonitor = false;
        }

        if(true == bStartMonitor)
        {
            /* Continue monitoring */
            R_IIC_SlaveMonitor(
                SLAVE_CHANNEL,
                PDL_NO_DATA,
                Rx_Buffer,
                RX_BUFFER_SIZE,
                slave_callback,
                7
            );
        }
    }

    /* The master has sent us data (now in the Rx_Buffer),
    store it in the data_storage array. */
    static void StoreData(uint16_t count)
    {
        uint16_t index = 0;

        /* Update data_storage_index */
        data_storage_index = Rx_Buffer[index];
        count--;
        index++;
```

```
        /* Store any data */
        while(count != 0)
        {
            data_storage[data_storage_index] = Rx_Buffer[index];
            count--;
            index++;
            data_storage_index++;
            if(data_storage_index == STORAGE_SIZE)
            {
                /* Wrap around */
                data_storage_index = 0;
            }
        }
    }
```

Figure 50: Virtual IIC Slave memory

5.20. Serial Peripheral Interface

5.20.1. Synchronous transfer with 32-bit data

This is an example of Serial Peripheral Interface usage where one SPI master communicates with one SPI slave.

The 2 separate RSK evaluation boards are used to connect the SPI together.

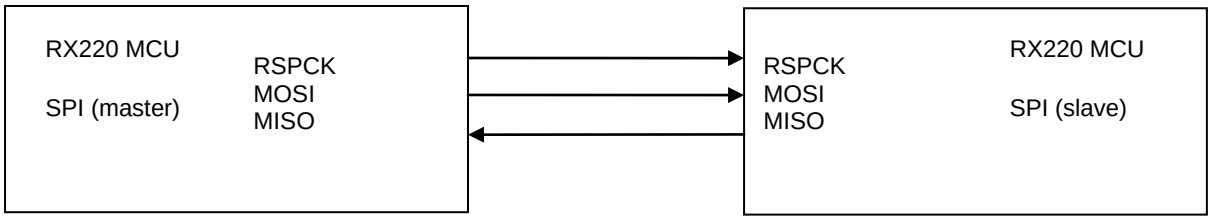


Figure 51 shows how four 32-bit words are transmitted and received simultaneously by the master.
The received data is then checked to confirm that the transfer was successful.

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

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

volatile bool master_transfer_complete;

void spi_master_callback(void);

#define MASTER_CHANNEL 0

void main(void)
{
    const uint32_t master_tx_data[4] = \
    {
        0x00000001,
        0x98765432,
        0xABCDEF34,
        0x12345678
    };

    const uint32_t slave_tx_data[4] = \
    {
        0x32323232,
        0x3456789A,
        0xDEADBEEF,
        0xFEEDCEDE
    };

    uint32_t master_rx_data[4] = \
    {
        0x00000000,
        0x00000000,
        0x00000000,
        0x00000000
    };

    uint8_t i;

    /* Initialise the system clocks */
    NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.   Please
refer to 5.1 Clock Generation Circuit.
```

```
    /* Configure the slave SPI IO pin */
    R_SPI_Set(
        PDL_SPI_RSPCKA_PA5 | PDL_SPI_MOSIA_PA6 | \
        PDL_SPI_MISOA_PA7
    );

    /* Configure the master SPI channel */
    R_SPI_Create(
        MASTER_CHANNEL,
        PDL_SPI_MODE_SYNC_MASTER,
        PDL_SPI_FRAME_1_4,
        PDL_NO_DATA,
        2E6
    );

    /* Configure the Master */
    R_SPI_Command(
        MASTER_CHANNEL,
        0,
        PDL_SPI_CLOCK_MODE_1 | PDL_SPI_LENGTH_32 | PDL_SPI_LSB_FIRST,
        PDL_NO_DATA
    );

    master_transfer_complete = false;

    /* Transfer all the data once */
    R_SPI_Transfer(
        MASTER_CHANNEL,
        PDL_NO_DATA,
        master_tx_data,
        master_rx_data,
        1,
        spi_master_callback,
        15
    );

    while (master_transfer_complete == false);

    for (i = 0; i < 4; i++)
    {
        /* Did the Master input match the Slave output? */
        if (master_rx_data[i] != slave_tx_data[i])
        {
            /* Handle the error */
            while(1);
        }
    }

    while(1);
}

void spi_master_callback(void)
{
    uint16_t StatusValue = 0;
    uint16_t Sequence_count;

    /* Read the master channel status */
    R_SPI_GetStatus(
        MASTER_CHANNEL,
        &StatusValue,
        &Sequence_count
    );

    /* No errors? */
    if ((StatusValue & 0x000Du) == 0x0u)
    {
        master_transfer_complete = true;
```

```
        }
        else
        {
            /* Handle the error */
            while(1);
        }
    }
```

Figure 51: Example of Serial Peripheral Interface Transfer of 32-bit Data by Master

Figure 52 shows how four 32-bit words are transmitted and received simultaneously by the slave.
The received data is then checked to confirm that the transfer was successful.

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

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

void spi_slave_callback(void);

volatile bool slave_transfer_complete;

#define SLAVE_CHANNEL 0

void main(void)
{
    const uint32_t master_tx_data[4] = \
    {
        0x00000001,
        0x98765432,
        0xABCDEF34,
        0x12345678
    };

    const uint32_t slave_tx_data[4] = \
    {
        0x32323232,
        0x3456789A,
        0xDEADBEEF,
        0xFEEDCEDE
    };
    uint32_t slave_rx_data[4] = \
    {
        0x00000000,
        0x00000000,
        0x00000000,
        0x00000000
    };

    uint8_t i;

    /* Initialise the system clocks */
    NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.   Please
refer to 5.1 Clock Generation Circuit.

    /* Configure the slave SPI IO pin */
    R_SPI_Set(
        PDL_SPI_RSPCKA_PA5 | PDL_SPI_MOSIA_PA6 | \
        PDL_SPI_MISOA_PA7
    );

    /* Configure the slave SPI channel */
    R_SPI_Create(
        SLAVE_CHANNEL,
        PDL_SPI_MODE_SYNC_SLAVE,
        PDL_SPI_FRAME_1_4,
        PDL_NO_DATA,
        PDL_NO_DATA
    );

    /* Configure the slave */
    R_SPI_Command(
        SLAVE_CHANNEL,
        0,
```

```
            PDL_SPI_CLOCK_MODE_1 | PDL_SPI_LENGTH_32 | PDL_SPI_LSB_FIRST,
            PDL_NO_DATA
        );

        /* Prepare the Slave for data transfer */
        R_SPI_Transfer(
            SLAVE_CHANNEL,
            PDL_NO_DATA,
            slave_tx_data,
            slave_rx_data,
            1,
            spi_slave_callback,
            15
        );

        slave_transfer_complete = false;

        while (slave_transfer_complete == false);

        for (i = 0; i < 4; i++)
        {
            /* Did the Master output match the Slave input? */
            if (master_tx_data[i] != slave_rx_data[i])
            {
                /* Handle the error */
                while(1);
            }
        }

        while(1);
    }

    void spi_slave_callback(void)
    {
        uint16_t StatusValue = 0;
        uint16_t Sequence_count;

        /* Read the slave channel status */
        R_SPI_GetStatus(
            SLAVE_CHANNEL,
            &StatusValue,
            &Sequence_count
        );

        /* No errors? */
        if ((StatusValue & 0x000Du) == 0x0u)
        {
            slave_transfer_complete = true;
        }
        else
        {
            /* Handle the error */
            while(1);
        }
    }
```

Figure 52: Example of Serial Peripheral Interface Transfer of 32-bit Data by Slave

5.20.2. Synchronous transfer with 8-bit data

Figure 53 shows how strings of 8-bit data are copied into 32-bit buffers, then transmitted and received simultaneously by the master.
The received data is then checked to confirm that the transfer was successful.

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

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

    /* Required for this example */
    #include <string.h>

    #define MASTER_CHANNEL 0
    #define BUFFER_LENGTH 25

    const char master_data_to_be_sent[] = "SPI master";
    const char  slave_data_to_be_sent[] = "SPI slave ";

    void main(void)
    {
        uint32_t master_tx_data[BUFFER_LENGTH];
        uint32_t master_rx_data[BUFFER_LENGTH];

        static uint8_t i;

        /* Initialise the system clocks */
        NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.   Please
    refer to 5.1 Clock Generation Circuit.


        /* Configure the slave SPI IO pin */
        R_SPI_Set(
            PDL_SPI_RSPCKA_PA5 | PDL_SPI_MOSIA_PA6 | \
            PDL_SPI_MISOA_PA7
        );

        /* Configure the master SPI channel */
        R_SPI_Create(
            MASTER_CHANNEL,
            PDL_SPI_MODE_SYNC_MASTER,
            PDL_SPI_FRAME_1_1,
            PDL_NO_DATA,
            2E6
        );

        /* Configure the Master */
        R_SPI_Command(
            MASTER_CHANNEL,
            0,
            PDL_SPI_CLOCK_MODE_1 | PDL_SPI_LENGTH_8 | PDL_SPI_LSB_FIRST,
            PDL_NO_DATA
        );

        /* Clear the receive buffers */
        for (i = 0; i < BUFFER_LENGTH; i++)
        {
            master_rx_data[i] = 0x00000000;
        }

        /* Copy the source data into the transmit buffers */
        for (i = 0; i < strlen(master_data_to_be_sent); i++)
        {
```

```
            master_tx_data[i] = (uint32_t)master_data_to_be_sent[i];
        }

        /* Transfer all the data once by polling */
        R_SPI_Transfer(
            MASTER_CHANNEL,
            PDL_NO_DATA,
            master_tx_data,
            master_rx_data,
            (uint16_t)strlen(master_data_to_be_sent),
            PDL_NO_FUNC,
            0
        );

        /* Wait 10 ms for data completely sent out */
        R_CMT_CreateOneShot(
            0,
            PDL_NO_DATA,
            10E-3,
            PDL_NO_FUNC,
            0
        );

        for (i = 0; i < strlen(slave_data_to_be_sent); i++)
        {
            /* Did the Master input match the Slave output? */
            if ( (uint8_t)master_rx_data[i] != slave_data_to_be_sent[i])
            {
                /* Handle the error */
                while(1);
            }
        }

        while(1);
    }
```

Figure 53: Example of Serial Peripheral Interface Transfer of 8-bit data by Master

Figure 54 shows how strings of 8-bit data are copied into 32-bit buffers, then transmitted and received simultaneously by the slave.
The received data is then checked to confirm that the transfer was successful.

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

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

    /* Required for this example */
    #include <string.h>

    void spi_slave_callback(void);

    volatile bool slave_transfer_complete = false;

    #define SLAVE_CHANNEL 0
    #define BUFFER_LENGTH 25

    const char master_data_to_be_sent[] = "SPI master";
    const char slave_data_to_be_sent[] = "SPI slave ";

    void main(void)
    {
        uint32_t slave_tx_data[BUFFER_LENGTH];
        uint32_t slave_rx_data[BUFFER_LENGTH];

        uint8_t i;

        /* Initialise the system clocks */
        NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.   Please
    refer to 5.1 Clock Generation Circuit.


        /* Configure the slave SPI IO pin */
        R_SPI_Set(
            PDL_SPI_RSPCKA_PA5 | PDL_SPI_MOSIA_PA6 | \
            PDL_SPI_MISOA_PA7
        );

        /* Configure the slave SPI channel */
        R_SPI_Create(
            SLAVE_CHANNEL,
            PDL_SPI_MODE_SYNC_SLAVE,
            PDL_SPI_FRAME_1_1,
            PDL_NO_DATA,
            2E6
        );

        /* Configure the slave */
        R_SPI_Command(
            SLAVE_CHANNEL,
            0,
            PDL_SPI_CLOCK_MODE_1 | PDL_SPI_LENGTH_8 | PDL_SPI_LSB_FIRST,
            PDL_NO_DATA
        );

        /* Clear the receive buffers */
        for (i = 0; i < BUFFER_LENGTH; i++)
        {
            slave_rx_data[i] = 0x00000000;
        }

        /* Copy the source data into the transmit buffers */
```

```
        for (i = 0; i < strlen(master_data_to_be_sent); i++)
        {
            slave_tx_data[i] = (uint32_t)slave_data_to_be_sent[i];
        }

        /* Prepare the Slave for data transfer */
        R_SPI_Transfer(
            SLAVE_CHANNEL,
            PDL_NO_DATA,
            slave_tx_data,
            slave_rx_data,
            (uint16_t)strlen(slave_data_to_be_sent),
            spi_slave_callback,
            15
        );

        while (slave_transfer_complete == false);

        for (i = 0; i < strlen(master_data_to_be_sent); i++)
        {
            /* Did the Master output match the Slave input? */
            if (master_data_to_be_sent[i] != (uint8_t)slave_rx_data[i])
            {
                /* Handle the error */
                while(1);
            }
        }

        while(1);
    }

    void spi_slave_callback(void)
    {
        uint16_t StatusValue = 0;
        uint16_t Sequence_count;

        /* Read the slave channel status */
        R_SPI_GetStatus(
            SLAVE_CHANNEL,
            &StatusValue,
            &Sequence_count
        );

        /* No errors? */
        if ((StatusValue & 0x000Du) == 0x0u)
        {
            slave_transfer_complete = true;
        }
        else
        {
            /* Handle the error */
            while(1);
        }
    }
```

Figure 54: Example of Serial Peripheral Interface Transfer of 8-bit Data by Slave

5.20.3. 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.

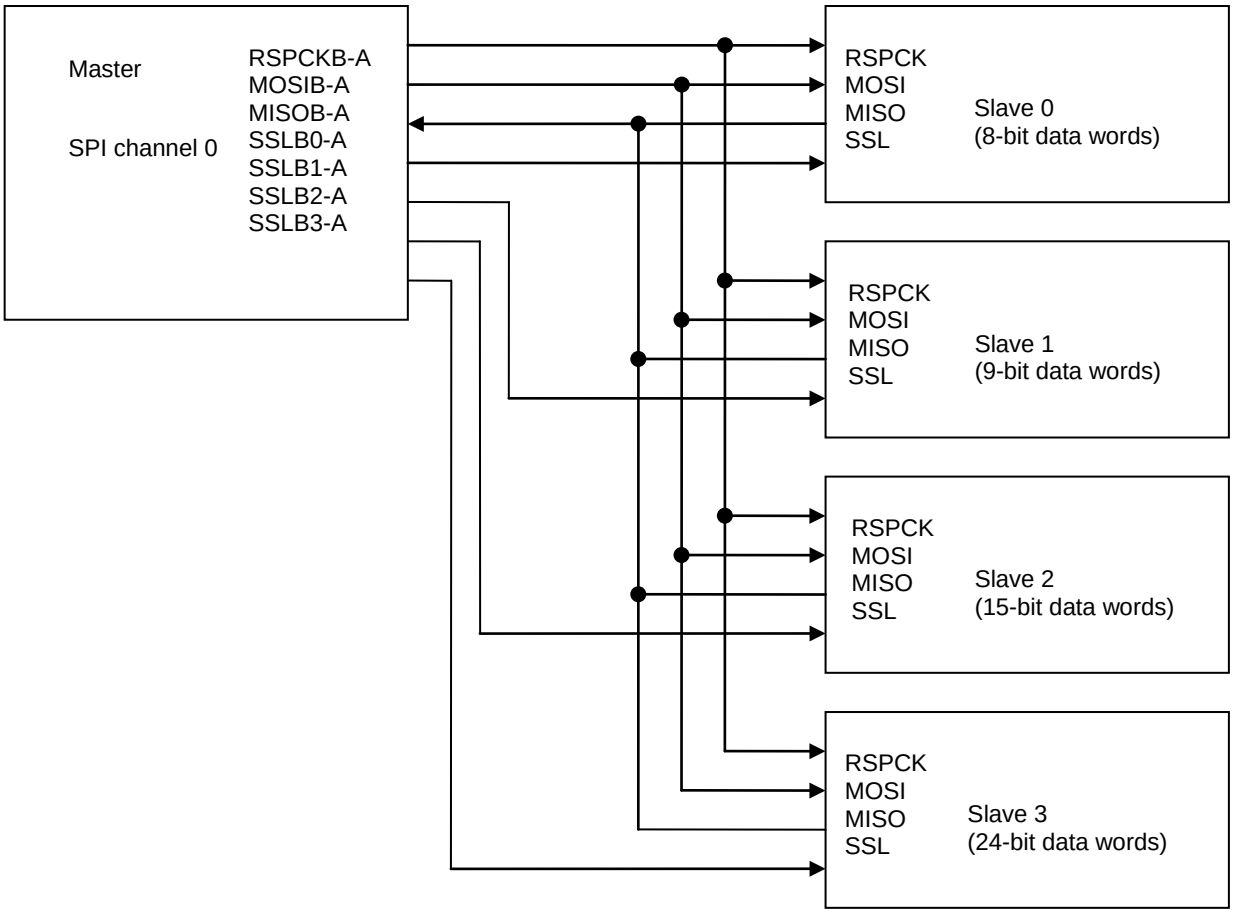


Figure 55 shows how data of appropriate bit lengths is transferred to each SPI slave.
Commands 0 to 3 are executed in sequence, with each command asserting the appropriate SSL pin.

```
/* Peripheral driver function prototypes */
#include "r_pdl_spi.h"
#include "r_pdl_cgc.h"
#include "r_pdl_cmt.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 */
        NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.   Please
    refer to 5.1 Clock Generation Circuit.

        /* Configure SPI Pin */
        R_SPI_Set(
            PDL_SPI_RSPCKA_PA5 | PDL_SPI_MOSIA_PA6 | \
            PDL_SPI_MISOA_PA7
        );

        /* Configure the master SPI channel */
        R_SPI_Create(
            MASTER_CHANNEL,
            PDL_SPI_MODE_SPI_MASTER | \
            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 55: Example of multiple slaves Serial Peripheral Interface use

5.21. CRC calculator

Figure 56 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"

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

void main(void)
{
    uint16_t crc_result;

    /* Configure the CRC to use the CCITT polynomial; */
    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; Expected result is 0 */
    R_CRC_Read(
        PDL_NO_DATA,
        &crc_result
    );

    /* Shutdown the CRC unit */
    R_CRC_Destroy(
    );
}
```

Figure 56: Example of CRC calculation

5.22. 12-bit Analog to Digital Converter

Figure 57 shows ADC_12 used in single scan mode, with a software trigger and a specified sampling time.

```
/* Peripheral driver function prototypes */
#include "r_pdl_adc_12.h"
#include "r_pdl_cgc.h"

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

/* Array used to read the ADC results */
uint16_t ADC_12_result[16];

void main(void)
{

    /* Initialise the system clocks */
    NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.   Please
refer to 5.1 Clock Generation Circuit.

    /* Configure analog input for AN015 */
    R_ADC_12_Set(
        PDL_ADC_12_PIN_AN015_PE7);

    /* Configure ADC for single scan */
    R_ADC_12_CreateUnit(
        0,
        PDL_ADC_12_SCAN_SINGLE | \
        PDL_ADC_12_ADSSTR_CALCULATE,
        PDL_NO_DATA,
        PDL_NO_DATA,
        5E-6,
        0,
        PDL_NO_FUNC,
        0,
        PDL_NO_FUNC,
        0);

    /* Configure ADC on AN015 */
    R_ADC_12_CreateChannel(
        0,
        15,
        PDL_NO_DATA,
        5E-6);

    /* Start ADC */
    R_ADC_12_Control(
        PDL_ADC_12_0_ON);

    /* Read ADC result */
    R_ADC_12_Read(
        0,
        ADC_12_result,
        PDL_NO_PTR);

    /* Shut down ADC */
    R_ADC_12_Destroy(
        0);
}
```

Figure 57: Example of ADC_12

5.23. Comparator A

Figure 58 shows an example of Comparator A usage.

```
.   /* Peripheral driver function prototypes */
    #include "r_pdl_cgc.h"
    #include "r_pdl_cpa.h"
    #include "r_pdl_intc.h"
    #include "r_pdl_io_port.h"

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

    void NMI_handler_cpa(void);
    void CPA0_handler(void);
    void CPA1_handler(void);

    uint8_t FlagsNonMASKABLE = false;

    void main(void)
    {
        uint8_t FlagsStatus;

        /* Initialise the system clocks */
        NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.   Please
    refer to 5.1 Clock Generation Circuit.

        R_IO_PORT_Set(PDL_IO_PORT_1_4, PDL_IO_PORT_OUTPUT);
        R_IO_PORT_Set(PDL_IO_PORT_1_5, PDL_IO_PORT_OUTPUT);
        R_IO_PORT_Set(PDL_IO_PORT_1_6, PDL_IO_PORT_OUTPUT);

        R_IO_PORT_Write(PDL_IO_PORT_1_4, 1);    /* off LED0 */
        R_IO_PORT_Write(PDL_IO_PORT_1_5, 0);    /* on LED1 */
        R_IO_PORT_Write(PDL_IO_PORT_1_6, 0);    /* on LED2 */

        /* Monitoring Comparison A channel 0 results */
        R_CPA_Create(
            0,
            PDL_NO_DATA,
            PDL_NO_FUNC,
            0
        );

        /* Get status LVDiDET LVDiMON */
        do
        {
            R_CPA_GetStatus(&FlagsStatus);
            if ((FlagsStatus & 0x01) == 0x01)
                break;

        }while(1);

        /* Comparator A channel 1 interrupts using Digital Filter */

        /* Enable the LOCO clock*/
        R_CGC_ControlAll(PDL_CGC_CLK_LOCO,PDL_CGC_LOCO_ENABLE,PDL_NO_DATA);

        /* Configure the NMI pin:  Non-Maskable Interrupt for Comparator A channel 1 */
        R_INTC_CreateExtInterrupt(
            PDL_INTC_NMI,
            PDL_INTC_FALLING|PDL_INTC_LVD2_ENABLE,
            NMI_handler_cpa,
            7
        );

        /* Non-Maskable Interrupt: Digital Filter, interrupt enable, LOCO_div_2 */
            R_CPA_Create(
```

```
            1,
            PDL_CPA_FILTER_LOCO_DIV_2 | PDL_CPA_INTERRUPT_RESET_ENABLE,
            PDL_NO_FUNC,
            0
        );

        FlagsNonMASKABLE = true;

        do
        {
            if (FlagsNonMASKABLE == false)
                break;

        }while(1);


        /* Disable comparator A channel 1 */
        R_CPA_Control(
            1,
            PDL_CPA_LVD_CIRCUIT_DISABLE
        );

        /* Maskable Interrupt: Digital Filter, interrupt enable, LOCO_div_2 */
        R_CPA_Create(
            1,
            PDL_CPA_FILTER_LOCO_DIV_2 | \
            PDL_CPA_INTERRUPT_RESET_ENABLE |\
            PDL_CPA_MASKABLE_INTERRUPT,
            CPA1_handler,
            6
        );

        do
        {
            /* Get status LVDiDET LVDiMON */
            R_CPA_GetStatus(&FlagsStatus);
        } while (1);
    }

    /* Comparator A channel 0 */
    void CPA0_handler(void)
    {
        uint8_t FlagsStatus;

        /* Get status LVDiDET LVDiMON */
        R_CPA_GetStatus(&FlagsStatus);

        /* Toggle the LED state */
        R_IO_PORT_Modify(PDL_IO_PORT_1_4, PDL_IO_PORT_XOR, 1);
        R_IO_PORT_Modify(PDL_IO_PORT_1_5, PDL_IO_PORT_XOR, 1);
    }

    /* Comparator A channel 1 */
    void CPA1_handler(void)
    {
        uint8_t FlagsStatus;

        /* Get status LVDiDET LVDiMON */
        R_CPA_GetStatus(&FlagsStatus);

        /* Toggle the LED state */
        R_IO_PORT_Modify(PDL_IO_PORT_1_4, PDL_IO_PORT_XOR, 1);
        R_IO_PORT_Modify(PDL_IO_PORT_1_6, PDL_IO_PORT_XOR, 1);
    }

    void NMI_handler_cpa(void)
    {
        uint8_t irq_status;
```

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

        FlagsNonMASKABLE = false;
    }
```

Figure 58: Example of Comparator A

5.24. Data Operation Circuit

This shows the configuration of the DOC and the DMAC to sum an array of numbers.

```
/* PDL functions */
#include "r_pdl_doc.h"
#include "r_pdl_dmac.h"

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

#define DMAC_CHANNEL     0
#define DATA_COUNT    10

extern void Callback_Done(void);

/* Data to calculate sum of. */
static uint16_t data[DATA_COUNT] = {1,2,3,4,5,6,7,8,9,10};
/* Callback Flag */
static volatile bool g_bCallbackDone = false;

void main(void)
{
    uint8_t status;
    uint16_t result;

    /* Initialise the system clocks */
    NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.

    /* Setup the DOC in addition mode, initial value = 0 */
    R_DOC_Create(
        PDL_DOC_MODE_ADD,
        0,
        PDL_NO_FUNC,
        0
    );

    /* Setup DMAC to write data to the 16bit DOC Input register */
    R_DMAC_Create(
        DMAC_CHANNEL,
        PDL_DMAC_BLOCK | PDL_DMAC_SOURCE_ADDRESS_PLUS | \
        PDL_DMAC_DESTINATION_ADDRESS_FIXED | PDL_DMAC_SIZE_16
|PDL_DMAC_IRQ_END,
        PDL_DMAC_TRIGGER_SW,
        data,             /* Source */
        (void*)&DOC.DODIR,     /* Destination */
        1,            /* Transfer Count */
        DATA_COUNT,        /* Data length */
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        Callback_Done,        /* Callback done function */
        7             /* Interrupt priority */
    );

    /* Enable and start the DMAC */
    R_DMAC_Control(
        DMAC_CHANNEL,
        PDL_DMAC_ENABLE | PDL_DMAC_START,
        PDL_NO_PTR,
        PDL_NO_PTR,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA
```

```
        );

        /* Wait for the DMAC to complete */
        while(false == g_bCallbackDone){;}

        /* Read the result including checking for overflow */
        R_DOC_Read(
            &status,
            &result
        );
    }
```

Figure 59: Example of DOC

5.25. Multi-Function Pin Controller

Figure 60 shows an example of Multi-Function Pin Controller usage.

```
/* Peripheral driver function prototypes */
#include "r_pdl_mpc.h"

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

void main(void)
{
    uint8_t data;

    /* Write data to register P07PFS */
    R_MPC_Write(
        PDL_MPC_REG_P07PFS,
        0xC5
    );

    /* Set bit 3 in P07PFS to 1 */
    R_MPC_Modify(
        PDL_MPC_REG_P07PFS,
        PDL_MPC_OR,
        0x08
    );

    /* Get the value of register P07PFS */
    R_MPC_Read(
        PDL_MPC_REG_P07PFS,
        &data
    );

    while(1);
}
```

Figure 60: Example of MPC

5.26. Bus Controller

Figure 61 shows an example of Bus Controller usage.

```
/* Peripheral driver function prototypes */
#include "r_pdl_bsc.h"
#include "r_pdl_io_port.h"
#include "r_pdl_cgc.h"

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

bool error_detected;

void BSC_error_handler(void);

void main(void)
{
    uint8_t status;
    uint16_t bad_address;
    volatile uint8_t temp;
    uint8_t * address_pointer;

    /* Toggle the priority to the Internal Peripheral Bus 1
      between Main Bus 1 and Main Bus 2. */
    R_BSC_Set(PDL_BSC_PRIORITY_PB1_MB1);

    /* Check priority register is as expected. */
    if(BSC.BUSPRI.WORD != BIT_4)
    {
        while(1);
    }

    /* Address that will generate an error */
    /* (See table "Types Of Bus Error") */
    address_pointer = ( uint8_t *)0x000C0000ul;

    /* Configure the bus controller */
    /* Enable illegal address detection and register a callback. */
    R_BSC_Create(
        PDL_BSC_ERROR_ILLEGAL_ADDRESS_ENABLE,
        BSC_error_handler,
        15
    );

    error_detected = false;

    /* Generate an illegal address error */
    *address_pointer = 0xAA;
    temp = *address_pointer;

    /* Wait for interrupt to set this. */
    while(error_detected == false);

    /* The interrupt handler should clear the detected status
    error, so read until it is clear. */
    /* Read the status registers */
    do{
        R_BSC_GetStatus(
            &status,
            &bad_address
        );
    }while(status != 0);

    while(1);
}
```

```
    void BSC_error_handler(void)
    {
        error_detected = true;

        /* Clear the error signals */
        R_BSC_Control(
            PDL_BSC_ERROR_CLEAR | PDL_BSC_DISABLE_BUSERR_IRQ
        );
    }
```

Figure 61: Example of BSC

6. RX-specific notes

6.1. Interrupts and processor mode

The RX CPU has two processor modes; supervisor and user.
The API driver functions may be executed by the CPU in either mode.
However, any callback functions which are called by the API interrupt handlers will always 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	RX220 Group User's Manual

Rev.	Date	Description	
		Page	Summary
1.00	Nov. 09, 2012	—	First Edition issued.
1.01	Feb. 07, 2013	4-50 4-119	R_CGC_Control: Added missing options. R_RWP_GetStatus: Change PDL_NO_DATA to PDL_NO_PTR.
1.02	Sep. 18, 2013	2 2 4 6 10 13 78 80	Add the "1.2. Compiler options when you use this product". Change 1.3.2 content into "Using RPDL stand-alone". Add content "Copy folder "\RPDL" into the folder project workspace created. (Example "C:\WorkSpace\rpdl_lib_test\ rpdl_lib_test")". Add content "To use library with debug information, enter "RPDL\RX220_library_debug" as the File path." Add content "In this section, only options which you must change from the default settings are described. If you add RPDL in existing project, see also "1.2 Compiler options when you use this product". Add section "11) Using library with debug information". R_IO_PORT_ReadControl: correct typo error "b7 – b1" to "b7 - b0" R_IO_PORT_ModifyControl: correct typo error "b7 – b1" to "b7 - b0"
1.10	Apr. 01, 2014	15 24 37 48 51 52 53	Add "Please refer the program in Section 5 Real-time Clock operation is not used and sub clock is not fitted on the board". MCU overview: Change to "Controlling the MCU features and on-chip ROM." RTC: Add "Reconfigure callback function and priority setting of alarm and periodic interrupts at warm start up" Table 2: Add R_RTC_CreateWarm R_CGC_Set: (a) Revise description of the sub-clock oscillator drive ability. (b) Add new parameter (data8) sub clock stabilization times (c) Delete "Options which are applicable only when the sub-clock oscillator is selected in parameter data1." (d) Add data8 description sub clock stabilization times R_CGC_Set: Revise remarks. ➢ Delete remark: If the sub-clock is selected, the Start type status flag will be set to warm (see R_MCU_GetStatus). ➢ Change description: Old : "Call this function once for each clock source that will be used" New: "Call this function once to set the clock frequency for each clock source whether it is used as system clock or RTC count source." ➢ Add remark: a)"Because this can not be done if ROM/Flash Program Erase mode is set or if a operating power mode transition is in progress, this function will return false if this is detected." b) When RTC is not to be used, call R_CGC_Control with option PDL_CGC_RTC_NOT_USE after calling this function to configure the RTC count source

Rev.	Date	Description	
		Page	Summary
			c) Make sure PCLKB clock frequency ≥ RTC count source clock frequency. d) Sub-clock oscillator is not available for 48 pin package.
		53	Revise the Program example
		54	R_CGC_Control: delete the options a) High-speed on-chip oscillator frequency control b) Main clock oscillator drive type 1 control c) Main clock oscillator drive type 2 control d) Sub-clock oscillator drive control
		55	R_CGC_Control: add the option “RTC initialization control.” R_CGC_Control: Revise remarks. ➢ Delete remark: If the sub-clock is selected, the Start type status flag will be set to warm (see R_MCU_GetStatus). ➢ Change description: Old:” Use R_CGC_Set to configure a clock source before selecting it.” New: “Use R_CGC_Set to configure a clock source before calling this function” ➢ Add remark: a) “This function can not be used if ROM/Flash Program Erase mode is set or if an operating power mode transition is in progress. This function will return false if this is detected.” b) Calling R_RTC_Create after using option PDL_CGC_RTC_TO_BE_USED. c) Make sure PCLKB clock frequency ≥ RTC count source clock frequency. d) Call R_CGC_Set once to set sub- clock frequency before call R_CGC_Control with option PDL_CGC_RTC_TO_BE_USE. e) Sub-clock oscillator is not available for 48 pin package
		92	R_MCU_Control: a) Remove On-chip RAM control b) Replace remark by “The PDL_MCU_WARM_START is used after the initialization of cold start (caused by a power-on reset) has completed. This is to indicate the next reset processing is warm start (Caused by a reset signal during operation).”
		99	R_LVD_Create: Add remark “User wants to use both LVD1 and LVD2: user must configure both LVD1 and LVD2 simultaneously”
		113	R_LPC_Control: Change PDL_LPC_SLEEP_RETURN_CHANGE_DISABLE to not default option.
		114	R_LPC_Control Remarks: Add “Do not try to change operating mode if ROM Program Erase mode is set or a mode transition is already in progress. This function will return false if this is detected.”
		149	R_ELC_Control: Add remark” Event PDL_ELC_LINK_EVENT_SPI_ERROR cannot be used if multi-master configuration, SPI operation, and master mode are selected for the RSPI. This function will return false if this condition is detected.”
		156	R_MTU2_Create: add “This option can be selected in other modes.” For ADC trigger control
		157	R_MTU2_Create: Revised restriction for buffer operation
		171	R_MTU2_ControlUnit: Revise description of compare match clearing control to

Rev.	Date	Description	
		Page	Summary
			"applies only to complementary PWM modes 1"
		172	R_MTU2_ControlUnit: Add remark for the condition of TCDR and TDDR
		215	R_RTC_Create : add new option Clock RTCOUT output period Select
		217	R_RTC_Create revise remarks ➢ Deleted remarks a) Before calling this function the sub-clock must be enabled and stable. Hence, use R_CGC_Set or R_CGC_Control to enable the sub-clock and then allow the clock stabilization time to pass before calling this function. b) If this function has been used and then a warm reset is performed it is not necessary to call this function again to continue using the RTC. However, if this function is to be called, it is necessary to call R_CGC_Set or R_CGC_Control to enable the sub-clock (even if it is already enabled) before calling this function. c) Call R_CGC_Set to set PCLKB clock frequency = sub-clock frequency when sub-clock is source clock. d) Call R_CGC_Set to set PCLKB clock frequency ≥ sub-clock frequency when sub-clock is not source clock. ➢ Add remarks a) Before calling this function the count source must be enabled and stable. Refer R_CGC_Set and R_CGC_Control for count source and stabilization time configuration. b) This function is called to use RTC after setting option PDL_CGC_RTC_TO_BE_USED in R_CGC_Control at cold start. c) This function is not required when using 48-pin package.
		218	R_RTC_CreateBinary : add new option Clock RTCOUT output period Select
		219	R_RTC_CreateBinary : revise remarks ➢ Deleted remarks a) Before calling this function the sub-clock must be enabled and stable. Hence, use R_CGC_Set or R_CGC_Control to enable the sub-clock and then allow the clock stabilization time to pass before calling this function. b) If this function has been used and then a warm reset is performed it is not necessary to call this function again to continue using the RTC. However, if this function is to be called, it is necessary to call R_CGC_Set or R_CGC_Control to enable the sub-clock (even if it is already enabled) before calling this function. c) Call R_CGC_Set to set PCLKB clock frequency = sub-clock frequency when sub-clock is source clock. d) Call R_CGC_Set to set PCLKB clock frequency ≥ sub-clock frequency when sub-clock is not source clock. ➢ Add remarks a) Before calling this function the count source must be enabled and stable. Refer R_CGC_Set and R_CGC_Control for count source and stabilization time configuration. b) This function is called to use RTC after setting option

Rev.	Date	Description	
		Page	Summary
			PDL_CGC_RTC_TO_BE_USED in R_CGC_Control at cold start. c) This function is not required when using 48-pin package.
		220	R_RTC_Destroy: add “This function is not required when using 48-pin package. “
		221	R_RTC_Control: add new option Clock RTCOUT output period Select
		223	R_RTC_Control : revise remarks ➢ Deleted remark: a) Call R_RTC_Create first before using this function b) If R_RTC_Create has been used and then a warm reset is performed it is not necessary to call R_RTC_Create again before using this function. However, it is necessary to call R_CGC_Control (or R_CGC_Set) to enable the sub-clock (even if it is already enabled) before calling this function. ➢ Add remarks a) This function is called after R_RTC_Create or R_RTC_CreateWarm. b) This function is not required when using 48-pin package.
		225	R_RTC_ControlBinary : add new option Clock RTCOUT output period Select
		226	R_RTC_ControlBinary : revise remarks ➢ Deleted remark: a) Call R_RTC_CreateBinary first before using this function b) If R_RTC_CreateBinary has been used and then a warm reset is performed it is not necessary to call R_RTC_CreateBinary again before using this function. However, it is necessary to call R_CGC_Control (or R_CGC_Set) to enable the sub-clock (even if it is already enabled) before calling this function. ➢ Add remarks a) This function is called after R_RTC_CreateBinary or R_RTC_CreateWarm. b) This function is not required when using 48-pin package.
		228	R_RTC_Read : ➢ Change remark: “Call R_RTC_Create or R_RTC_CreateWarm first before using this function” ➢ Add remark: “This function is not required when using 48-pin package”
		229	R_RTC_ReadBinary: ➢ Change remark: “Call R_RTC_CreateBinary or R_RTC_CreateWarm first before using this function” ➢ Add remark: “This function is not required when using 48-pin package”
		230	R_RTC_ReadBinary: Add Program example
		231	R_RTC_CreateWarm : new API added
		232	R_IWDT_Set: ➢ Add remark “The IWDT counter frequency must not be greater than the PCLB / 4. Set the IWDTCLK division ratio accordingly. This function will return false if this condition is detected” ➢ Delete remark “Call R_CGC_Set to set PCLKB clock frequency >= 4 times IWDTCLK clock frequency after division”
		237-238	R_SCI_Set: Add pins selection for channel 9, 12
		248-249	R_SCI_Receive: Add continuous receive mode option and remarks

Rev.	Date	Description	
		Page	Summary
		269	R_IIC_MasterSend : ➢ Add option PDL_IIC_10_BIT_SLAVE_ADDRESS.
		270	➢ Remove remark “Channels 1 and 3 are not available with the 80-pin and 100-pin packages. This function will return false in this case.”
		271	R_IIC_MasterReceive: ➢ Add option PDL_IIC_10_BIT_SLAVE_ADDRESS.
		272	➢ Remove remark “Channels 1 and 3 are not available with the 80-pin and 100-pin packages. This function will return false in this case.”
		273,275, 276,277, 278	Remove remark in R_IIC_MasterReceiveLast, R_IIC_SlaveMonitor, R_IIC_SlaveSend, R_IIC_Control, R_IIC_GetStatus: “Channels 1 and 3 are not available with the 80-pin and 100-pin packages. This function will return false in this case.”
		300	R_ADC_12_CreateUnit : Add remark
		301	R_ADC_12_CreateChannel: Add remark
		304	R_ADC_12_Control: Add remark “Do not select CPU Off unless there is any interrupt to wake up the CPU.”
		305	R_ADC_12_Read: Add remark for self-diagnosis result format
		307	R_CPA_Create: Revise example
		318-427	Add cross-reference for chapter 5
		319-320	INTC: Update usage example
		321-322	I/O PORT: Update usage example
		323	MCU: Update usage example
		326-327	CAC: Update usage example
		333-336	DTC: Update usage example
		341-342	MTU2: Add sample code for Reset-Synchronized PWM mode
		343-344	TMR: Update usage example
		347-367	RTC: Update usage example
		370-390	SCI: Update usage example
		409-412	SPI: Update usage example
		420-422	CPA: Revise sample code
		425	Add usage example for MPC
		426-427	Add usage example for BSC
1.11	Aug. 01, 2014	149	R_ELC_Control: Revised remark: Old: “Event PDL_ELC_LINK_EVENT_SPI_ERROR cannot be used if multi-master configuration, SPI operation, and master mode are selected for the RSPI. This function will return false if this condition is detected.” New: “Event PDL_ELC_LINK_EVENT_SPI_ERROR cannot be used if multi-master configuration, SPI operation, and master mode are selected for the RSPI.”

Renesas Peripheral Driver Library
User's Manual
RX220 Group

Publication Date: Rev.1.11 Aug 01, 2014

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.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

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

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

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

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

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Ku, Seoul, 135-920, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 3.0

RX220 Group

Renesas Electronics Corporation

R01US0059EG0111